

BABEȘ-BOLYAI UNIVERSITY CLUJ-NAPOCA
FACULTY OF MATHEMATICS AND COMPUTER
SCIENCE

SPECIALIZATION Computer Science in English

DIPLOMA THESIS

**Representationally sane user
interfaces on the dual spaces of
quasicrystals**

Supervisor
Lector Bența Kuderna-Iulian

Author
Bodică Septimiu-Călin

2022

ABSTRACT

User interfaces are generally developed using specification languages that have a hierarchical nature. An alternative is to use reflection and rotation groups such as Coxeter groups to inductively define a geometric structure. To validate alternative approaches to defining layouts, a method of collecting biometric data to study the user's interaction with the user interface is proposed. Justifications are provided for design choices and regarding which algebraic groups are most well suited for human-computer interaction. A framework is developed to study user interaction through hand drawing tasks which are linked to the biometric data and geometric structures representing the tasks. The framework is developed using structured software engineering practices and can be extended upon to satisfy any concrete use case. The application is generalized through using an information retrieval interface that can be replaced to extend to any possible use case.

Contents

1	Introduction	1
1.1	Background and motivation	1
1.2	Human-computer interaction	3
2	Related work	8
2.1	Signal processing	9
2.2	Stroke embeddings	10
2.3	Artificial neural network models	11
3	Methods	14
3.1	Problem model	14
3.1.1	Overview	14
3.1.2	Mathematical description	15
3.1.3	The state machine correspondence	17
3.1.4	Sequences of predictions	20
3.1.5	Complexity of the tilings	21
3.2	Tilings	22
3.2.1	Background	22
3.2.2	Mathematical background	23
3.2.3	Motivation	24
3.3	Tiling grammar	24
3.4	Feature extraction	26
3.4.1	Homologous representation using the linear Hough transform	27
3.4.2	Image retrieval	29
4	Application development	31
4.1	Requirements	31
4.2	Analysis and design	36
4.2.1	Dependencies and structure	36
4.2.2	Motivation for software choices and alternatives	37
4.3	User interface and game window	39

4.4	Database	40
4.4.1	Data analysis	41
5	Human-computer interaction and virtual reality	44
5.1	Measurements related to brain activity	44
5.1.1	Background	44
5.2	Data and software	47
5.3	Data and code availability	48
6	Conclusions	49
6.1	Future work	50

Chapter 1

Introduction

1.1 Background and motivation

Computer software are sometimes complex, requiring an elaborate specification and code to perform even simple tasks. Depending on the programmer's preferences and biases, different approaches may be taken when solving a problem. A programmer's understanding of another programmer's solution may be different than the original author's expectation, and an author's own work may seem alien to themselves when the work is taken out of context. Working in large, collaborative software projects involves the skill of reading multiple styles of code, which can vary substantially from person to person. The process of peer review is based on sharing different perspectives about the software design process, and over time can generate consensus in the form of standards, conventions and best practices. One such example is the Python Enhancement Proposals (PEP), which are based on the principles recorded in the "PEP 20" document. These standards evolve slowly over time as the community gains and shares experience gained by using the capabilities of the programming language, and reach consensus.

Code naturally goes through a process of compilation, because for a human it is much easier to work with representations resembling natural language. The compiler accepts the natural language and produces machine-interpretable code automatically. A result of compilation is that regardless of subjective differences in implementation, the same outcome is achieved, by forcing the users to respect key constraints in the language specification. The constraints are enforced by introducing compilation errors, warnings, suggestions.

Linters are extensions that programmers can install in their IDE to statically analyze code style and improve the chances of a correct outcome. They work independently from the compiler but can still improve the programmer's yield, with the advantage that the linter is maintained independently of the compiler. A set of

separable code evaluation and validation methods, such as the compiler and linter, is more valuable than a single centralized component which may have many cross-cutting concerns and too many responsibilities, deviating from the SOLID principles. Compilers are complex tools that change slowly. A linter can be updated regularly with the help of community driven feedback. There are many such linters for the language Python, which enforce the PEP standards.

Compilers can reduce multiple solutions to the same program of machine code approaching a single ultimate representation, by going through optimization and rewriting processes. This abstract final representation is hard for humans to interpret. While the compilation process has a single resulting machine code corresponding to the original solution, working backwards from assembly or machine code to high level programming languages is not a tractable problem because of the large search space. Other methods of writing good natured code is by following functional programming paradigms and compose functions using the *Typed Lambda calculus*, or by delegating the responsibility of composing aspects of an application or class to the compiler, as seen in aspect oriented programming languages.

A fully fledged application can have many interacting components, which need to be configured properly so that no unpredictable behaviour can occur. Sometimes, however, these individual components have large similarities and even shared logic between them. As such having a compressed, readable, well-formed, predictable source code is desirable. The purpose of this study is to present a framework for profiling user actions. The framework attempts to generalize the process of harnessing signals from a user interface across multiple possible input sources, separating aspects of any potential GUI into a maximal set of independent parameters, following the previously mentioned good practices.

In the first chapters, a mathematical basis for discussing the general aspects of user interfaces is proposed. The software created for this study is a meta-user-interface. The purpose is not to interface with any existing layout specification technology or connect to the user interface of an application that is already available, but to provide a mathematical context for analyzing the use of user interfaces. The goal is simply to discuss the mathematical properties of user interfaces and provide a base for building analyses of user interfaces. Of particular interest are the representation of the meta-user-interface application's components, including the data structures employed. The data structures used in the software solution are following the mathematical description and specification discussed in Chapter 3, and are designed in accordance with software development lifecycle principles like Agile and DevOps. The goal is to identify what duplicate problems exist across seemingly separate aspects of the UI with the goal of achieving a minimal representation of UI actions. Questions could be what are the most reused components, what are the

cross-cutting concerns, what are the unnecessary duplications? The framework proposes a way of matching the user input to the UI's response, as well as the user's response to events visible on the GUI, through creating a mathematical context compatible with the field of human computer interaction.

The second part is found after Chapter 3. It opens a discussion about performing tasks such as correlating externally collected data about the user, biometric data, sensor fusion of biometric modalities, muscle tracking, and others, and using this plenitude of data to select valid hypotheses about the experience of the user, for the purpose of designing UIs that satisfy the largest number of possible users.

The final goal of this framework is to draw conclusions about a possible correlation between the underlying mathematical description of the perceived environment of a user, and the user's conscious and unconscious ability to perceive the rich environment, or the user's failure thereof. The application provides a context for discussing human-computer interaction (HCI), an research topic that can be improved by the availability of big data and latest neural network designs. By contrasting the human brain with neural networks, the objective is to gain insights into what elements can be eliminated and what alternative representations there are that achieve equally good results to classical UI design patterns. Neural networks are able to learn different types of sensing features than the human brain, which has evolved to solve its own set of tasks. Some features learned by the deep layers of neural networks are still interpretable, and can be insightful if properly visualized. In the following chapters, there will also be discussions of how neural networks can interface with particular aspects that any user interface is comprised of.

The benefit of connecting HCI to representation theory is the ability to translate the conclusions into already existing user interfaces. One advantage in this pursuit is that a UI has an underlying specification that is often provided as a nested hierarchy, a graph, or an XML. Finding a correlation between a user's profile and a specification language can then be used to adapt UI behaviour to the optimal choice. If this solution is provably sub-optimal, the parameters can be slowly updated so that learning to transition to the desired outcome is as comfortable as possible, as seen by performing analysis and prediction on the collected UI interaction metadata. The user's natural instinct to categorize regions of a UI by their positioning and layout is tackled by matching the biometric data with the mathematical description that generates the corresponding layouts.

1.2 Human-computer interaction

Human-computer interaction is concerned with the similarities and dissimilarities between the human and the computer. The two are viewed as systems that commu-

nicate and mutually exchange information. Lately, neural networks and AI systems have grown significantly. This work is written with the consideration that the human brain and a neural network can both interact with a classical UI with the same degrees of success and they can be used interchangeably. Testing user interfaces with neural networks is somewhat simpler than a robotics task, where physical simulations about the environment replace the user interface. As discussed in the later chapters, there is an intersection between the physics of random processes on a large scale and the user behavior on the user interface. A way to approach problems that have high complexity is to reduce the search space through choosing a local region of interest within which a solution can be found, after which the solution is used to describe the problem globally.

The data gathered about the usage of the UI can be processed and interpreted in order to gain knowledge about specific usage patterns, inefficiencies, and use the knowledge to design and improve systems. It is closely related with the fields of neuroscience, computer science, physiology, network dynamics, and others. An overlap may also be seen between HCI and BCI, the emerging fields of virtual reality, augmented reality and extended reality (AR, XR), and distributed systems. The reason for this is that increasingly large quantities of data are being viable for inference by edge-devices that are part of and form larger cloud-native structures. The user interfaces may see a transition to a more human embodiment, based on the scientific field of integrated information theory. [1] Complex queries may be formed by combining recommendation engines with other input modalities such as brain waves, voice input, body tracking, eye tracking, and personalized insights.

Combining multiple streams of data from different sensing domains of a system can improve the quality of each individual signal, increase predictability of a system, and decrease the susceptibility and resilience to subsystem failure. Common methods for sensor fusion include voting mechanisms or combining multiple agent's sensing to achieve coordinated swarm behavior. [2] User feedback can be considered an example of a complex swarm behavior, as seen in large online recommendation systems that work using neural networks. [3, 4]

Biological systems are highly complex systems. There are many measurements that can be taken, and the explanations for the change in a signal often involve interactions of a large number of subsystems of which not every aspect can be monitored in real time. Often but not always the amount of data that can be collected is much larger than what is needed to achieve good results. UIs generally follow principles of simple designs because they are easy to understand, which implies that the data that can be collected about using a UI is simple, it can be compressed or reduced without losing the essential information that it contains such as transitions or committing an action. Sometimes, additional measurements can be made,

or one wants to consider how the usage patterns of a large number of users might be interpreted. As will be shown in the discussions, forming hypotheses about the reasons for which random variables or joint probabilities might be correlated is complex. Sometimes, automated theorem provers, or decision support systems, or proof assistants, are the best tools to navigate the complexity of the human body's interconnected measurable signals, as human language will probably not be useful.

Measurements that have use potentials include heart rate, blood pressure, electrodermal response, and EEG, and they all could be used to analyze the body's reaction to various tasks. In creative tasks, such as drawing, sculpting, writing, EEG has already been studied. EEG has correlated spatial and temporal information from the signals detected from the scalp depending on the nature of the task. [5] Recent progress in the field of virtual reality (VR), show that eye tracking might be an accessible source of valuable signals. The headsets in the future could incorporate EEG, and some commercially available wearables already come with well documented methods of using their collected data.

Electrocorticography approaches the ideal scale for measuring the highly complex human brain. For a long time only optical imaging or magnetic sensing was achievable, but projections show that research groups such as Neuralink can increase the resolution with advanced materials. [6] While EEG is not as spatially precise as electrocorticography, choosing the right number of electrodes and sampling frequency can provide enough statistical evidence for localizing a significant amount of brain activity that responds to a perceived stimulus when performing a task. If the task has a high compatibility with the scalp measurements, the cross correlations between the channels can be used as a brain-computer interface (BCI). Computational models using statistical techniques for feature extraction together with machine learning models are methods used to understand biometric data streams, but visualization and data understanding are important to explain and justify a model's performance. The human body is too complex and dependent on personal characteristics and a solution that works might not always be explainable.

Recently BCI have been used with more success for communicating with disabled patients in the form of neurofeedback systems that co-adapt with a learning patient in order to use raw brain activity with the help of sound stimuli to produce natural language text. [7] Because of the structured nature of brain connectivity, electric activity measures can be interpreted differently based on the location where they occur. Some theories about how signals propagate through the brain are compatible with the concept of receptive fields from the study and design of artificial neural networks (ANN). In the case of visual perception, the raw optical input arrives in the hindbrain, or occipital area, and gets diffused along the temporal and central axes, eventually having the potential of reaching the frontal lobe.

For the receptive fields of the visual cortex, one could expect the electrodes T3, T4, T5, T6 to activate for handwriting because they are located close to the temporal lobe, and more predominant C3, CZ, C4 activity for drawing faces, with the most raw input being found in the occipital region (Fig. 1.1). The apparent justification for such a hypothesis is that the cortical homunculus associated to the facial nerves has been identified to be closer to the central region of the electrode map. [8] At the same time, both of these regions are in proximity to regions associated with high order visual areas. Thus the signals are dependent on the sensory stimuli. The signal's localization can also depend on the patient's profile, and personal characteristics. The problem is that the space of hypotheses to test with respect to the neural signals is very large. There could be problems with associating signals from the temporal lobe to those on the central lobe because there might be hidden variables such as obstructed or unmeasured regions on which the signals depend. For this reason a decision support system or proof assistant may be better suited to probe the realm of possibilities. [9, 10]

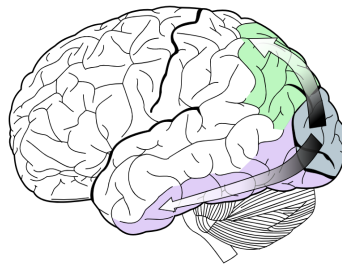


Figure 1.1: Illustration of the ventral-dorsal stream hypothesis. Source: based on a figure available freely in Henry Gray (1918) *Anatomy of the Human Body*, modified by Wikipedia user Selket.

In the second part of study the framework presented is used to determine if data from a hand drawing task can be used to understand more about the user. An attempt is made to make the input compatible with recommendation systems, or make a profile based on the user's preference of the UI layout. The evolution of the user's state could then be used to predict future behavioral patterns, and if possible to guide the user into a favorable state by changing the parameters of the UI. It is observed that there is a similarity factor between hand drawing tasks and using a cursor to interact with a 2-dimensional GUI.

The framework is able to use data about UI usage in order to personalize the UX, or user experience. The identified parameters are the main tools for this end goal. It is not enough to have the parameters alone, but the choice of parameters must be justified and based in mathematically provable facts. In the methods section a description of the mathematical model shows that a complexity measure is a valu-

able tool for identifying the features of a UI. A large body of mathematics focused on representation theory is referenced. By connecting representation theory to user interfaces, an enriched discussion of user interfaces, feature extractors, gesture detection and finally data structures is presented.

Chapter 2

Related work

Familiarity with the usual software engineering concepts, data structures, artificial neural network glossary are useful, as well as the literature covering general aspects of what was mentioned in Chapter 2. Most of the work presented in the framework aims to provide the context for discussing UI concepts, and ideas to improve designs. To do this, mathematical language is used that is found in linear algebra, group theory, geometry. Because the environment in which the application runs is a Turing machine built on *von Neumann* architecture, there is an implicit mathematical toolset for discussing the complexity and computability of the application. Formal methods, automata theory, static analysis may be seen as related to the nature of the tilings presented in later sections. The language generating the tilings, groups, have a similar nature to some formal grammars, following strict expansion rules. Such formal methods have a problem when applying them to the highly nonlinear and complex human brain, and even some larger neural network models. Even the mathematical models described in Chapter 3 should only outline a problem that is intrinsically very complex.

To illustrate a potential bridge between the simple mathematical description and the complicated physics involved in HCI, consider the classical problem of the brain-computer interface, which has been a subject of research that dates back as far as the 1930s. A BCI has to solve a multitude of tasks: abnormality detection, signal classification, pattern detection and predictions on a certain window, generating synthetic signals and interpreting the data with respect to other sensor modalities, such as heart rate or muscle movement. To solve all of these tasks at once, the analog signals must be converted to digital ones and stored. Using a computer with an operating system has the advantage that the solution provided can be event-driven, and human-readable data structures can be used in specifying the program's behavior. The events and data structures have a nested or hierarchical representations, because some of the tasks are inherently subordinate to others. This tends to form a so-called *pipeline*, where the raw signal is transformed successively and branched

into parallel processing steps. For example, to classify a multi-channel EEG signal, one would take the raw signal, obtain a compressed digital signal at a sampling rate ω , obtain the STFT (Short-time Fourier transform) which is a spectrogram, feed the image into a computer vision machine learning model, and output the confidence of each prediction, given by the softmax function:

$$\sigma(x_i) = \frac{e^{x_i}}{\sum_{j=1}^K e^{x_j}},$$

which is a linear activation function. The softmax function is an example of a function that maps probabilities to a discrete set of outputs based on the relative difference between the weights corresponding to each class.

This example is shown to highlight the fact that a neural network consists of linear activation functions as well as nonlinear ones. In the exemplified pipeline, most transformations can be considered to be linear operations, with the exception of the fully connected neural layers of the ANN. Connecting multiple fully connected layers of neurons of a neuron network together is a process that produces behavior that is considered highly nonlinear and difficult to predict. Furthermore, the human agent can be considered a part in this pipeline, or part of a more complex human-in-the-loop system. The only way to cope with the complexity of nonlinear systems is to add constraints to either their input, which is the interface with the environment, or constrain the behavior of the system instead, like limiting the number of output classes. Adding constraints enables some properties to be derived and allows the systems to even be simulated to a small degree.

Other scenarios exist using different techniques and are detailed in the next subsection. To conclude the introduction to this chapter, the purpose of the thesis is reiterated: to obtain a correspondence between highly deterministic aspects of user interfaces, and highly complex and nonlinear domains such as the human brain.

2.1 Signal processing

To detect abnormalities, the signals are analyzed in sequences of batches of a certain length of time. Statistics are extracted at the granular level of individual time windows, or the correlations between them can be considered instead. The field of digital signal processing includes fields such as spectral analysis, Fourier analysis, and the study of nonlinear systems. By comparison to digital signals, analog signals are continuous in the time domain. An analog signal can be very precise up to a certain level where the noise is inseparable from the signal changes.

Sometimes specialized hardware is used to convert the analog signal which is obtained from the mechanical or electrical sensor, into a digital signal, without delegat-

ing this responsibility to the computer, using an analog-to-digital converter (ADC). The inverse task is performed with a digital-to-analog converter, but in the processing pipeline we are interested in the digital signal. If the instrument possesses an ADC, then the signal that is obtained from the instrument is already split into discrete values at a certain sampling rate ω measured in Hz. The discrete set of values can be reconstructed from the samples using numerical approximations. Some spectral or compressed representations can also be used by machine models. Alternative representations of a signal include: STFT (Short-time Fourier transform), wavelet transform. The wavelet transform can be used to define a Hilbert basis. Because of the Hilbert space's unique properties it has the ability to compress a large quantity of EEG signals into orthonormal wavelets, which makes it easier to conceptualize and visualize as well as use them with machine learning models. [11]

For predicting signals, there is a problem when having to choose the right time window, and correlate the environmental factors with the signal to obtain a probability of correctness associated with the prediction. Imagine that the raw signal is used as input in a prediction pipeline, the prediction of the next 5 second interval might not only depend on the previous 5 seconds, but also on statistical indicators that are correlated with the perceived environment. As habituation occurs the signals could change as a function of the biological factors or personal traits that are unique to each individual, which gives rise to the dimension of cross-subject statistical indicators. Since this unfolds in the time domain, it might be considered a highly complex dynamical system. Even a pipeline of hand-crafted features could contain errors outside a narrow set of constraints in which the system can be used. A small neural network has been shown to produce results similar to differential equation solver, so one could assume that using a complex multi-layer ANN can produce a good prediction of the time series signal and how it depends on the input variables, in a similar fashion to the manually crafted feature extraction pipeline. The problem then becomes understanding the justification of the prediction, which is likely to be an abstract list of neurons weights with no interpretable meaning outside of the neural network itself. The function that the neural equation solves might not always be linear or continuous as is preferred for mathematical models. The following section discusses the problems raised by using layered neural networks for signal processing tasks.

2.2 Stroke embeddings

Compositional stroke embeddings (CoSE) are a way of representing hand drawings as a series of pen strokes, and interpret them using transformers and self-attention based neural networks. [12] These paradigms of neural network architecture design

are succinctly explained in the following subsection.

The discreteness of the events represented by brush strokes enable RNNs to make predictions about the next set of strokes. It has been shown in other language-related RNN studies, that the possibility of breaking a problem down from word-level analysis to character-level analysis can yield better representations of the latent space and also allow phonetic structures to be considered. This level of granularity has given an advantage compared to other models when applied to some cross-lingual neural machine translation tasks. [13, 14]

The CoSE neural network model uses a dataset of diagrams to make a prediction about the next components of the diagrams. This makes CoSE fall under the classification of a generative model. CoSE is a complement to the other architectures that solve tasks related to handwriting, language and text. The latent space of the model is closer to the bottleneck section of the model and its positioning and dimension is what gives it the ability to generalize toward previously unseen data. In the study, one of the aims is to analyze the potential of optimizing the latent space by using the body of mathematics known as representation theory. As seen in the introduction, the claim is that the way of generating periodic and aperiodic tilings is not straightforward, and serves as a good starting point for defining complexity, information entropy, and generalizability. In fact some problems as the Domino Problem (see Wang dominoes) are closely related to decidability and undecidability problems, and the halting problem of Turing machines.

The discrete events understandable by a recurrent neural network provide bounds within which the analysis of the data can take place. When working with geometric structures like tilings of the Euclidean plane, the previous data that was drawn has to be accounted for when trying to understand the next steps. Recurrent neural networks can model complex probabilities without having the need to understand the probability functions describing them, given that the model performs well on evaluation to previously unseen data. In a large framework such as the one found in this work, the neural network models don't directly determine the next steps in a prediction sub-task, but can be part of a voting mechanism consisting of multiple neural network outputs, or a multi-head neural network model.

2.3 Artificial neural network models

Neural network models can achieve better results than hand-crafted pipelines by being able to internally represent long term temporal dynamics. RNN and LSTM architectures can be used to predict a future time window. An RNN or recurrent neural network is a layered neural network which can accept sequential input and is capable of integrating data about the positioning and context of individual items

from the input sequence. The RNN architecture is very popular for natural language tasks and has had some success with time series data. LSTM or long short-term memory is built on RNNs but improve their contextualizing abilities by using a specific wiring between the layers called the attention architecture that improves language understanding. One problem with using such models on time series data is that the inputs can be extremely long. The data on which the RNN and LSTM models are usually trained are not large bodies of text, unless it is an experimental model like GPT-3 which can handle large texts but has a much more complicated architecture and takes more computing resources to train. For example, a 5000 word book when scanned at a sampling rate of 256Hz, it would take only 19 seconds or 0.3 minutes to parse. GPT-2 and GPT-3 accept roughly 2048 words as inputs, and have billions of parameters that need to be trained. As a result of the model complexity, it is able to solve many sub-tasks such as named entity recognition and translation. The weights propagate from layer to layer through the network when during, and it is possible that the individual units that solve these sub-tasks improve the performance of the other tasks as well.

This can raise the question of localizing neural activations in a neural network. The gradient of activations can depend on the input and is dependent on the output layers of the neural network. [15] There are tools that can analyze the activation gradients since every aspect of each neuron is completely known to the computer. For the human brain it is more difficult, and the task falls to instruments such as fMRI, photometric techniques like BOLD imaging, and EEG. LORETA is a standardized process for the localization of electrical activity in the brain from only measurements of electrical signals. It is possible to reason about event related potentials by understanding the functional connectivity of the brain regions assisted by a tractography. [16] As mentioned in the introduction, forming a hypothesis about what is correlated and what is not can be nearly impossible with natural language. Techniques such as probabilistic graphical models, combined with proof assistant systems are better suited for creating theories and complex logic. These systems take as input a set of constraints and use the measurements to produce a theorem based on those constraints, with techniques such as linear programming. Anumachipalli et al. have done research on using electrocorticography (ECoG) signals in the ventral sensorimotor cortex, superior temporal gyrus and inferior frontal gyrus with RNNs to translate imagined speech. [7] The sensorymotor cortex encodes the muscle activations of face at the syllable level in different locations. The ECoG sensors have enough spatial resolution to distinguish the coding of the imagined speech. ECoG have a higher spatial resolution than EEG for localizing brain activity, but its disadvantage is that it involves an invasive method.

Convolutional neural network (CNN) architectures are among the oldest and

most widely known. CNNs have been used with high success for predicting epilepsy or depression. [17, 18] These architectures take a 1-dimensional tensor as input, but it is also possible to consider classifying spectrograms of the signals, maintaining a high evaluation score. Such a spectrogram is an n -dimensional input, where n is the number of channels. [19] RNNs and CNNs still have a very large number of parameters, and they also have problems with small datasets and generalizing.

Other neural network models include generative models, which solve the task of generating synthetic signals of a specific class that could closely resemble real recordings. A generative adversarial neural network (GAN) is trained by using a so-called discriminator. The model takes as input an image from a dataset, and during training tries to match the output with the input. During inference, only vague information about the desired image is provided, and the network uses the learned aspects of the dataset to synthesize new, unseen outputs, that are indistinguishable from real samples. Because signals are just 1-dimensional samples, they can be formatted as an image, or instead their spectral representation can be provided. The GAN architectures are closely related to the encoder-decoder paradigm of neural network architecture design. They can do tasks similar to GANs, but instead of generating a signal, they use the learned features for classification. The representation of some models are just the inverted topology of other ones, or sub-modules connected together. The GAN has a dual bottleneck architecture. Their representation could prove insightful in the later discussions about representation theory of this study. [20, 21, 22] These architectures have higher potential of being able to generalize away from the limitations of a dataset while still maintaining valid results.

Autoencoder based architectures can learn deep features from a signal. A powerful set of emerging neural network architectures named graph neural networks allow deeper insights into what kind of features the networks learn. They allow better human interpretability of the learned representation, as explained by Demir et al. in the EEG-GNN work. [23]

Chapter 3

Methods

This chapter is structured in 3 parts. First, the environment and tools are described. Then, a mathematical description of the problem is given. Then, the mathematical description is referenced to make arguments about software-related aspects. The gap between neural networks and the human brain is outlined with remarks about the measures of complexity of the user interface using representation theory. The concept of latent space of a neural network is very useful in relating the two worlds. Finally some extensions are suggested, and some results are provided about using feature extractors and neural networks to aid this pursuit. The framework presented should be thought of as a human-in-the-loop ecosystem that has no intention of being complete but should be a tool based in mathematically sound descriptions that can be extended.

3.1 Problem model

3.1.1 Overview

The main piece of the application is the drawing board. This can be considered as the main component because of the fundamental way in which drawing captures kinematic data. The user's unique way of representing objects through hand drawing has been already studied, and different user profiles such as ones based on geographic region have already been documented. There is ample data to support building hypothesis about this topic. First of all, handwriting data itself is a kind of hand drawing task. Then, languages are already localized in geographic clusters. It was hinted already in Chapter 2 that drawing of diagrams is also related to handwriting in its own particular way. A neural network's representation of a dataset of hand drawn diagrams is very similar to handwriting because individual strokes compose larger constructs. What is omitted is the phonetic aspects of speaking a language, which goes to show that humans represent concepts such as

languages by integrating separate perspectives, like speaking, writing words, writing sentences, reading sentences into one whole. This is even more interesting when using languages composed of syllabic scripts mixed with multiple writing systems with numerous homographs, like Japanese and Kanji.

A drawing is composed of strokes, which is the interval of time between the moment when the pen is activated, and deactivated. Basic kinematic data comes with this such as velocity and acceleration. The shape drawn in a single stroke may be simple, such as a line, or complex, such as a set of curves or splines. Other dimensions of the stroke are the colour used, the thickness and shape of the brush, the number of cursors, the orientations of the cursors, the symmetry axes of the cursors and their relative positions.

Let there also be another category of information about the strokes that is called second-order characteristics. These are as follows: order of the strokes, family of slopes, time of initiation of a stroke, and the relationship of a stroke to past and future strokes. Relationship between strokes is expanded on within our framework by using a recommender system for a future stroke. The recommender comes with different heuristics for proposing a new stroke, with special interest on forming self-similar patterns, or following a probability distribution based on the length of a sequence of recommendation, for example.

3.1.2 Mathematical description

First the data structures of importance throughout the discussion are specified.

The objective of the task is the tiling, composed of edges:

$$L = \{(p_i, p_j) \mid i, j \in \mathbb{N}, p \in P_c\}. \quad (3.1)$$

Denote the related set P_c of points within the tiling edges:

$$P_c = \{(s_x, s_y) \mid s \in \mathbb{R}\}, \quad (3.2)$$

such that

$$\forall l = (p_i, p_j) \in L, p_{ij} \in P_c. \quad (3.3)$$

A distinct set from P_c is P_s which is all points that can be sampled on the canvas.

$$P_s = \{(s_x, s_y) \mid s \in \mathbb{R}\}, \quad (3.4)$$

with $P_c \subset P_s$.

The current task is denoted as:

$$D = \{\forall s \in P_{c,s}\}. \quad (3.5)$$

It should be noted that the input of two distinct entities or game actors are involved in the final result: the user and the game engine. While the resulting state is a composite of P_c and P_s , both need to be stored separately by the game engine, for a few reasons. First of all, inputs could overlap, in fact it is the user's objective in certain game modes to perfectly draw over the game's input. To measure the similarity of the player's input and the optimal game engine's suggestion, the sources of the two sample points need to be distinguished.

This consideration is important when generalizing to multiple agents or decomposing the game engine's input into multiple recommendation styles.

Note that more or less, the points $s \in D$ correspond to pixels or paint strokes on the whiteboard.

In equation (3.4) we defined P_s which is a very general space of all points that can be sampled. We expand this into the set of strokes H_s that can be collected, where H stands for *history*, which ultimately belongs in the state machine. Let

$$H_s = \{\forall s = \{p_i, \dots, p_j \mid p \in P_s\}\}, \quad (3.6)$$

with each stroke $s \in H_s$ and define the map,

$$\theta : \mathbb{N} \longrightarrow \mathbb{R}^2, \quad (3.7)$$

which maps all points from the stroke onto an interpolation function f :

$$\theta(p_{i,j}) = f(i) \mid i < j \in \mathbb{N}, \quad p \in s, \quad s \in H_s \quad (3.8)$$

s.t.,

$$\forall p_i \in s, \quad f(i) = p. \quad (3.9)$$

The following discussion is structured as follows: we describe the naturally occurring order structure induced by the temporal domain of the measurements. We then define a Push-Down Automata (PDA) and a set of symbols belonging to its associated grammar $G = \{N, \Sigma, P, S\}$, which describes the *finite state machine* of the system. The symbol N describes the set of states, Σ the allowed symbols on the input tape, P the productions describing state transitions, and S the final states.

First of all, let:

$$t : X \longrightarrow \mathbb{R}, \quad (3.10)$$

be the time function, satisfying:

$$L, P, H \subset X. \quad (3.11)$$

We then have the natural order structure:

$$t(s_i) \leq t(s_j). \quad (3.12)$$

A unique property given by the ability of a state to have multiple cursors nC is given by the potential equality,

$$t(s_i) = t(s_j) \iff nC > 1. \quad (3.13)$$

3.1.3 The state machine correspondence

The system is modelled by a push-down automata. Let M be a PDA:

$$M = \{Q, \Sigma, \Gamma, \delta, q_0, H_0, F\}, \quad (3.14)$$

with a set of states

$$Q = \{Init_i, Gen_i, Pred_i, Free_i \mid i \in \mathbb{N}\}. \quad (3.15)$$

The symbol Σ is the alphabet of symbols accepted on the input tape, Γ is the set of symbols accepted by the stack, δ the production rules that describe the transitions triggered by input symbols including stack behavior, q_0 the initial state, H_0 the initial stack symbols, and F the final states. These give the general classes of states but need not be complete in this description, because each individual state could be further decomposed and classified bringing little value to the discussion. Here, $Init_i$ is a class of initialization states when starting the program, Gen_i is a set of states used for generating the tiling objective, generating a new prediction or set of predictions, $Free_i$ is a cluster of states where the main mode of interacting is hand-drawing.

The PDA, M , is capable of switching game modes by parsing the input and triggering switches using the stack(s). The complexity of the inputs has no theoretical upper bound, and it will be shown that gestures can be integrated into the UI to eliminate buttons, which supports the study's pursuit of compressed representations. A further "complexification" is to include an image retrieval engine that uses features of a gesture to change the interface. Due to time constraints we don't fully implement the game mechanics of all game modes such as that of UI simulation, but only detail on the currently existing hierarchical representation of UIs in Chapter 5. Hypothetically, a Sim_i class of states would be introduced to account for this game

mode.

To model all possible events within the PDA, as well as signals, the alphabet for the input is the following:

$$\Sigma = \{Point_{\delta xy}, K_{\uparrow}, K_{\leftrightarrow}\}. \quad (3.16)$$

The associated set of rules δ of M are a set,

$$\delta : Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \longrightarrow Q \times \Gamma, \quad (3.17)$$

which makes use of the stack to implement switching logic as between game modes $q_i \in Q$, store the history of inputs H , and adjust the random variables of the probabilistic sampling χ_i such as when changing the number of predicted strokes and the frequency of the predicted strokes, or using gestures.

The final states F are also determined by gestures, or the probability of successfully completing a current task based on previous input, which is a final state specific to the current game mode only $F' \subset F$.

Consider drawing the edges of a tiling as game mode (1) and clicking buttons on a UI the game mode (2). The events involved in both are $Point_{\delta xy}$. The rules δ , depending on the state $q \in Q$, will either record a stroke when holding the mouse button K_{LMBdn} or releasing it K_{LMBup} . Ideally the system would capture data about all previous movements of the cursor, but when applying this on real hardware, the system would quickly run out of memory, especially when the cursor moves in n dimensions, or equivalently $Point_{\delta x_{\overline{1,n}}}, \langle x_i \rangle \in \mathbb{R}^n$.

Storing all history is not necessary when we record only the intervals delimited by the K_{LMB} inputs. The resulting state modifiers are recorded by the stack, and the inputs are inserted in the set H . Mathematically we denote a map between the grammar and the problem sets:

$$\begin{aligned} H_s, H_e &\in H \\ \phi^\dagger : G &\longrightarrow H_s \end{aligned} \quad (3.18)$$

$$\phi^\dagger(x_k, x_{k+1+n}) = (p_k, \dots, p_{k+1+n}, K_{\uparrow}), p \in P_s, K \in \Sigma \iff k_{+2+n} is K_{\uparrow} \quad (3.19)$$

A stroke will always begin with a vertex and terminate with a vertex. The dual problems of the tiling drawing and clicking on the UI elements is then as follows:

$$\phi^\dagger(x_k, x_{k+1}) = (p_{k-1}, K_{\uparrow}), p \in P_s, K \in \Sigma \quad (3.20)$$

while still maintaining the ability to interpolate two points

$$\delta : H_s^2 \longrightarrow H_s$$

$$\delta(s_i, s_j) = \begin{cases} y - y_{s_i} - m * (x - x_{s_i}) = 0 \\ m = \frac{y_2 - y_1}{x_2 - x_1} \end{cases} \quad (3.21)$$

Consider $l \in L$ a possible prediction with $L \subset P_c \times P_c$. We would like to impose an order structure:

$$\tau_1 : l_1, l_2, \dots, l_n$$

$$\Psi : L \times L \longrightarrow \{T/F\}, \quad (3.22)$$

$$\Psi(l_1, l_2) = m_{l_1} > m_{l_2},$$

where:

$$m_l = \frac{y_{p2} - y_{p1}}{x_{p2} - x_{p1}}, \quad p_{1,2} \in l,$$

and want to look for alternatives of Ψ where the comparison operator itself is a function:

$$g : L \times L \longrightarrow \{T/F\}, \Psi(l_1, l_2) = g(l_1, l_2).$$

For the inputs K_{\uparrow} and K_{\rightarrow} the state machine transitions into a batch prediction mode where multiple predictions are made. A prediction is a function of the order structure together with 2 random variables that describe the number k of strokes to predict, and the batches in relation to the time in which they will be provided.

$$P(\chi) = \begin{cases} 1, & \iff q = q_{full}, q \in Q \\ \frac{\lambda^k \cdot e^{-k}}{k!}, & \iff q = q_{timed}, q \in Q \end{cases} \quad (3.23)$$

where q_{timed} has a chance to give a prediction batch in a set of time intervals, but will always roll one of those intervals.

Remark that the number of cursors nC , the order structure corresponding to their orientations τ_v and the map of their relative positions δ_v are all characteristics of the next prediction synthesized by the program. This information is calculated on the moment of providing this prediction as-needed, and not at the initialization phase. Therefore there is a map,

$$\phi : Q \times \Gamma \longrightarrow P \times \tau, \quad (3.24)$$

where the tuple-set,

$$\tau = \{(nC, \tau_v, \delta_v, l) \mid l \in L\}, \quad (3.25)$$

is the set of possible current predictions mapped by ϕ .

Alternatively one should choose a mapping co-domain of ϕ s.t. additional information that was left out is included. For example a coloring of each vertex by relative position could result in a more complex sampling of the current prediction:

$$\phi : Q \times \Gamma \longrightarrow P \times \tau \times \gamma, \quad (3.26)$$

where $\gamma : N \longrightarrow \mathbb{N}^3, \gamma(x) = \langle i, j, k \rangle$ and $i, j, k \leq 255$.

3.1.4 Sequences of predictions

Recall the mapping ϕ (3.18) and let this generate the set $S = \{(nC, \dots, l), l \in L\}$. Take a sequence of predictions from this set $s_1, s_2, \dots, s_n, s \in S$. We have already defined an order structure of the orientations τ_v , for each individual stroke of this prediction, formally this means $(\tau_v, l) \subset s, s \in S$.

Further we can derive additional transformations through subsequent mappings. Let

$$\phi^1 = \{s_i \mid \overline{xy}_{j, s_{i-1}} = \overline{xy}_{i, s_i}, \forall i \in \mathbb{N}, x_{s_i} \in S \cap L\} \quad (3.27)$$

be the set resulting from mapping the set of all predictions by imposing the constraint that the endpoints of the lines be contiguous.

Theoretically there could be any number of such transformations. A single additional mapping that might be of interest is provided, ϕ^2 , in which the longest common subsequence (LCS) algorithm is used to introduce two additional tunable parameters: k and r , which are the chosen subsequence length and the number of repetitions.

$$\begin{aligned} \phi^2(k, r) &= f(g(S^1)) \\ &= f(s_i \mid s_i \in \phi^1(S)) \\ &= \{s_i \mid m_{l_{s_i}} = m_{l_{s_{i+r}}}\} \end{aligned} \quad (3.28)$$

having $\|S^3\| = m$, where m_i is the slope of the line.

Then, the problem has a set of mappings ϕ^i , that satisfy the constraints C:

$$C = \begin{cases} c_1, & \text{The endpoints are connected.} \\ c_2, & \text{They are formed by repetitions of the LCS.} \end{cases} \quad (3.29)$$

3.1.5 Complexity of the tilings

Let $Sp(X)$ be an operator acting on a set, named the spectrum of that set, and consider $L \supset P_c$. Then $Sp(L)$ is our measure of complexity, in the following way:

$$\begin{aligned} Sp(X) &= \langle m_i \rangle, \forall i \in \mathbb{N}, \\ &= \langle m'_i \mid \{-1, 1\} \rangle \end{aligned} \quad (3.30)$$

with $m'_i = |m_i|$ is the absolute value of the slope m_i . This goes to show that the number of reflection, rotation and transformation operations that produce the tilings can still generate finite entropy of information. The inner product of the linear space of slopes with the set $\{-1, 1\}$ produces the negatives of each slope, but this is only relevant when having to consider orientation, as we did in the mapping τ_v .

In the case of periodic or regular tilings, $Sp(X)$ would be enough, but for aperiodic tilings like Penrose tilings, quasicrystals, and quasi-periodic crystallographic groups, the following is proposed:

$$\max_{k \geq 2, r > 0} |\phi^2(k, r)|_{k,r} \quad (3.31)$$

which is to say find the k and r of the LCS function that maximize the value. Since we can't make claims about which of k and r is more important, the reader might either consider them separately or come up with a function $g : \mathbb{N}^2 \rightarrow \mathbb{R}$ to best suit their needs.

Note that one might also see a similarity between this measure and the span of a set $span(X)$, which might prove to be similar, but if one considers span of quasistructures, the previously defined Sp would be modified to include the LCS complexity measure. An idea to further refine it is to determine the probability χ of finding a value close to the $|\phi^2|_{k,r}$ in less than r' neighbors.

3.2 Tilings

3.2.1 Background

The representation of tilings is a complex topic. Usually there is a diagrammatic language or a sequence of symbols to denote a tiling. We refer to the sequences as notations, which are further described.

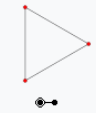




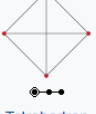
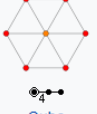
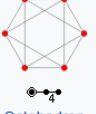


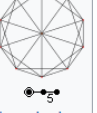
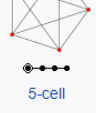
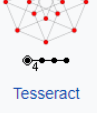
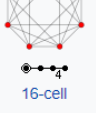
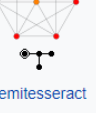

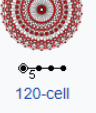
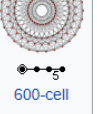
Table of irreducible polytope families [hide]									
Family n	n -simplex	n -hypercube	n -orthoplex	n -demihypercube	1_{k2}	2_{k1}	k_{21}	pentagonal polytope	
Group	A_n	B_n		$I_2(p)$ D_n	E_6	E_7	E_8	F_4	G_2
2	 Triangle	 Square		 p-gon (example: p=7)	 Hexagon			 Pentagon	
3	 Tetrahedron	 Cube	 Octahedron	 Tetrahedron				 Dodecahedron	 Icosahedron
4	 5-cell	 Tesseract	 16-cell	 Demitesseract	 24-cell			 120-cell	 600-cell

Figure 3.1: Picture of the diagrammatic notation. The Coxeter group diagrams are generated by a Lua script written by user Tomrueen which is part of Wikipedia's templating system.

The fundamental building block is a polygon, or sometimes the vertices and edges of these polygons. They can be generalized to n dimensions but for this discussion we refer to the Euclidean plane. Regular tilings involve regular polygons as the building blocks. There are specific operations done iteratively to form a composite of the fundamental blocks. These can be rotation and reflection operations, for example. A construction where every polygon has a neighboring one at every edge is called a tessellation.

It is possible that some notations are too restrictive to produce every possible case. Cundy and Rollet's notation has some problems regarding the ability to generate unique tessellations because of ambiguity. [24]

There are many classifications of the possible ways of tiling the Euclidean plane, and this has been a subject of scientific debate for decades, before 1987 when Grünbaum and Shephard published their work. [25].

To every tiling corresponds its dual representation, and possibly coloring together with other properties and particularities. Considering the dual graph where

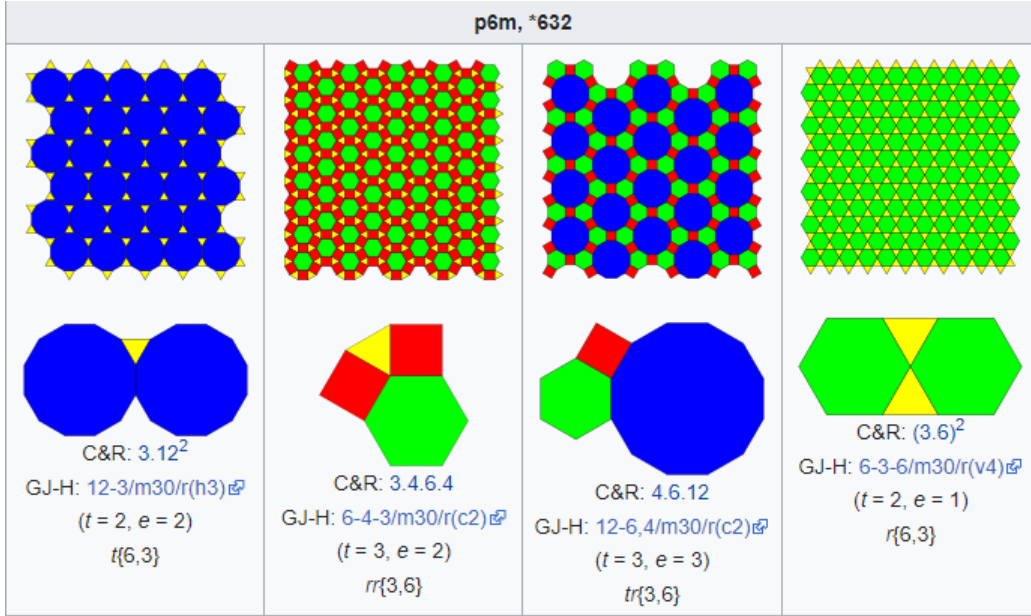


Figure 3.2: Picture of some plane filling tilings using regular polygons. Depicted examples are classified as demiregular tilings. Work of Wikipedia user Tomruen, using the KaleidoTile app for iOS and shared using a creative commons license.

the centroid of each polygon is a vertex, and these nodes are connected. When the resulting tiling is identical to the original one, the object is called self-dual.

Closely related figures are ones produced by the plane crystallographic groups, plane symmetry groups, or wallpaper groups.

3.2.2 Mathematical background

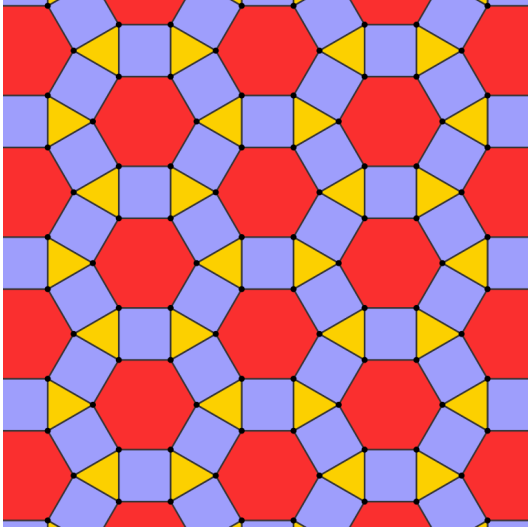
A Coxeter group is an abstract group that admits a formal description in terms of reflections.

The group has the following presentation: $\langle r_1, r_2, \dots, r_n \mid (r_i r_j)^{m_{ij}} = 1 \rangle$ where $m_{ii} = 1$ and $m_{ij} \geq 2$ for $i \neq j$. A presentation is a way of specifying a group from a set of generators and a set of relations. The relations in this descriptions are of the form $(r_i r_j)^{m_{ij}}$.

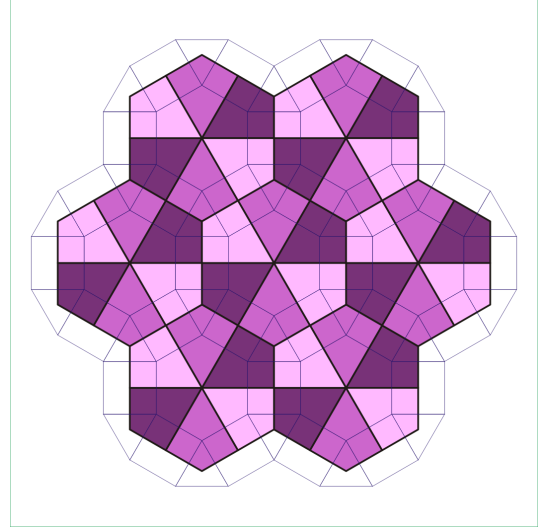
Ultimately one obtains a correspondence between a Coxeter diagram and a Coxeter matrix. For the formal details refer to Coxeter 1935. [26]

The Schläfli symbol has a notation of the form $\{p, q, r, \dots\}$ and can define regular polytopes and tessellations. For example, $\{3\}$ is an equilateral triangle, $\{4\}$ a square, and so on.

By using a fraction one can also describe pentagrams $\{\frac{5}{2}\}$, pentagons $\{\frac{5}{1}\}$, which are given by the number of vertices and their turning number. The notation has some useful properties, for example finding the dual of polytopes. If a polytope of dimension $n \geq 2$ has Schläfli symbol $\{p_1, p_2, \dots, p_{n-1}\}$ then its dual has the symbol



(a) The rhombitrihexagonal tiling.



(b) The deltoidal trihexagonal tiling, made by connecting the centroids of the rhombitrihexagonal tiling.

Figure 3.3: Picture of the rhombitrihexagonal tiling and its dual representation. Shared on the Wikimedia database by Wikipedia users: TED-43, Watchduck (Tilman Piesk) using a creative commons license.

$$\{p_{n-1}, p_{n-2}, \dots, p_1\}.$$

3.2.3 Motivation

Although the framework does not generate the tilings by itself, but instead uses feature extractors together with an Support-Vector-Graphics (SVG) parser, the mathematical context is considered very important to the study. It might prove valuable to attempt to compress a hierarchical description of a UI to its generator. Usually the UI is described by a tree structure such as an XML, XAML, or equivalent representation. It might be worth investigating if using symmetry groups and compositions of functions that do similar transformations can produce similar results to classical specification methods. It could also be investigated if the tree hierarchy of an UI, or more generally the scene graph of any virtual environment, can be decomposed into pieces that can be described by such special groups, and the graph might be annotated with this information.

3.3 Tiling grammar

Tiling grammars are proposed especially for a subset of tiling tasks where the regular polytopes don't have to fill the Euclidean plane or \mathbb{E}^3 space. The single constraint is that the length of the connecting faces must be the same. The application

is that a sequence of the grammar's symbol can be generated simply from cursor movements, and by operating on with predefined rules the grammar's symbols, the resulting composition might be rewritten or compressed.

It is up to the creator of the systems how they will use the formal grammar, this study's intent is to merely outline the language that can be used. Some ideas about how it might be applied are: combining the tiling problem on any of the dual graphs of tilings to produce UI transitions by pattern matching, navigating semantic nodes, and using the feature detector to match previous compositions by drawing a simple curve that matches the trajectory. It is possible to interpolate the drawn curve to a Bézier curve or another type of interpolated spline, which also normalizes it and removes noise, and use the curve to match the edges on the dual space of the history of tiled compositions. Graphically the size of a tiling composition can be adjusted to please the user and provide feedback at every step of the way, but abstractly the representation is as simple as a b-SPLINE and uses almost no memory.

Using either the spline, a tiling composition, or a gesture, can embed a large quantity of information by matching with the history of user interaction or using semantic networks or concept maps coupled to an information retrieval engine. It could be possible to construct a JSON representation of the search constraints and, provided an API exists at the UI layer for interpreting it, a transition can be made to the correct activity by pattern matching on the JSON constraints, while also being able to specify the desired destination layout. For the sake of discussion name this API the *Graph query* API. The reader can find more about this in the concluding remarks, Chapter 6. The Graph query would theoretically allow constraint programming using the geometric compositions that reflect the search space in real time during the construction of the query by providing feedback to the user.

Suppose the left grammar,

$$\begin{aligned} G &= \{N, \Sigma, P, S\}, \\ \Sigma &= \{i \in \mathbb{N}\}, \end{aligned} \tag{3.32}$$

and an automaton that accepts pairs of two symbols X_k, X_{k+1} of input length $2 \cdot n$. An input sequence that looks like 0, 6, 1, 3, 2, 4 will first produce a central hexagon, then attach a triangle on edge 1, and a square on edge 2 of the triangle. There are already formal definitions of such processes, but this example is shown because the programmer has the freedom to design any such equivalence in order to produce a set of tilings.

Following, each symbol is expanded with a set of symbols from another grammar,

$$G' = \{N', \Sigma', P', S'\}, \tag{3.33}$$

and the input $\forall X_i \in \Sigma$ is rewritten to :

$$\begin{aligned} X_j'' &\in \{0X_1, 6X_2, 1X_3, \dots X_i X_j' \mid X_i \in G, X_j' \in G'\} \\ X_j'' &\in G' \cup G. \end{aligned} \tag{3.34}$$

The symbols X_j'' are equivalent to a decision support feedback process that is given to the user at the insertion of each symbol in the input tape. An insertion of the symbol corresponds to one of the dual transitions ϕ^\dagger defined in equation (3.16), which is paired to our now-defined grammar of walks on edges. The final point is coupling the decision support symbols X_j'' to a data structure such as a graph. Any system that traverses the vertices of the figures generated can be viewed as a graph. The graph of transitions on the geometrical structure can transition between walks on edges, and walks on vertices of the figures, as long as the grammar allows it. Such a graph of vertices generated by centroids, edges or vertices, or any mapping of the components of the tilings to a set of points, can belong to a symmetry group like the Coxeter groups. Depending on the number of options available during a walk for the next transition, the system can decide the number of faces of the adjacent polygon as the tiling is generated. A transition can be viewed as adding a constraint to the space of possible polygons connected to the current one as the tiling is generated. Some of the faces can be constrained in order to not limit the queries by future self-collisions, or to compose polygons so that the construction will finish by running out of options in a number of steps. Ultimately, these compounds can be logged and rewritten or searched by pattern matching, or a feature search.

3.4 Feature extraction

The mathematical notation in the previous sections included the notion of spectrum of a set $spec(X)$ and the modified version $Sp(X)$. This section provides a discussion about them, and the related concepts of feature extraction, for two applications. First, to connect the complexity of the tiling to the LCS algorithm using a feature extraction technique. At the same time, the complexity of the tiling's representation is discussed, for both periodic and quasi-periodic tilings. Secondly, the feature extractor is applied to a gesture detection module that is integrated in the application's lifecycle and directly supports the human-in-the-loop system. Gesture detection is shown to be generalizable to multiple input types that can include eye tracking and other pen-related information when using digital tablets such as pen angle and pressure. Both eye tracking and digital pen tablets provide kinematic information or information that can be transformed into kinematic information which the framework is specifically designed for. Using the language of rotation and re-

flection groups, the 2D canvas could be extended in the future to include 3D shape reasoning, as the group that describes a 2-dimensional composition of polygons can also describe a 3-dimensional figure belonging to the same group.

The framework permits incorporating biological signals such as EEG data to correlate the mathematical structures to evoked responses within the human brain. Raw EEG signal manipulation provides a search space that is too complex, and the aim of the framework is to reduce the search space using the well defined geometrical properties of tilings. A discussion about biometric data for knowledge mining using graphical models is related and discussed in Chapter 5, which also involves connectograms and probabilistic graphical models for creating hypotheses.

The motivation for suggesting to use a feature extractor combined with an information retrieval system is that it serves as a starting point for the game engine's recommendation system. The game engine can be seen as a normal user, having the same degrees of freedom with respect to input on the drawing board. The game engine should be aware of the local as well as global context of what was previously drawn. Local features that evolve during a drawing session might be influenced or constrained by neighboring objects represented within the scene. With respect to the human in the loop system, the user might also be unconsciously influenced by what is perceived in the periphery of the focus of attention and could be something to investigate. Information retrieval systems can associate similar inputs to particular results that have the objective of satisfying the user's intent, and therefore an analysis of information retrieval systems is recommended using the features produced by the drawing because it implies that there are facts to be discovered about drawing tendencies.

3.4.1 Homologous representation using the linear Hough transform

The Hough transform is a feature extraction technique that is used to detect specific shapes like sinusoidal curves or lines. Instead of describing a line by $y = mx + b$, it is transformed as a function of distance from a fixed origin and the angle between the x axis and a line connecting the origin to that point, or:

$$r = x \cdot \cos(\theta) + y \cdot \sin(\theta) \quad (3.35)$$

Then each line appears as a pair (r, θ) in the new plane, also known as the Hough space.

Because of the fixed origin, and the relation between (r, θ) and an individual line, and the fact that the *regular* tilings are composed of lines, the Hough transform can be used to discuss particular aspects of a tiling, especially reasoning about local regions of a large composition of polygons. Using a converter written in

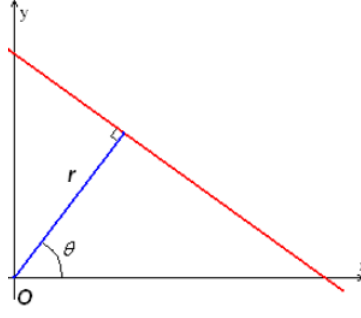


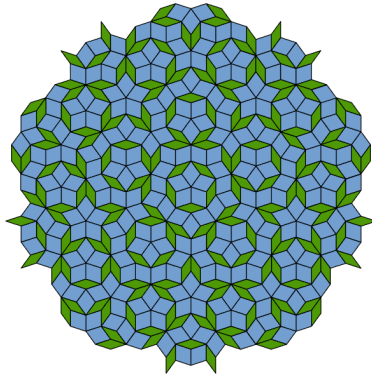
Figure 3.4: Illustration of the Hough space. Source: image generated and uploaded by user Shcha on the Wikimedia database.

Python, a series of experiments determined that a set of tilings involving a dodecagon, hexagons, octagons, squares and triangles, and even one quasi-periodic Penrose tiling, all had the same transform on a large resolution picture of the tiling. This could be caused by inadequate resolution of the transform and the closeness of the edges with respect to the angle θ causing the numerical values to vanish at a large radius r . The situation is different when changing the origin to be on the points of the dual graph of a tiling, and choosing a small r . It was found that a different transform appeared for every r within one repetition of neighbor polygons of the one which has the centroid in the origin O .

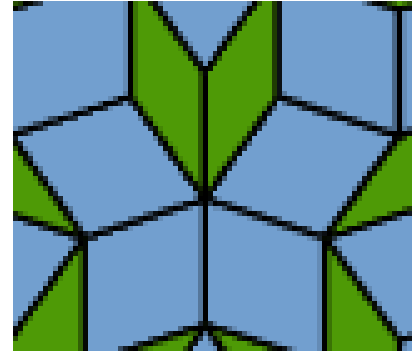
Since data about the slope is already incorporated in the parameter θ , this opens an alternative way of comparing two neighbourhoods of a centroid in a tiling, other than the ones previously described. In the LCS algorithm in Section X, the walks were also defined on the vertices of the polygons, not on the dual space. This procedure gives a numerically viable way of representing $Sp(X)$ as $H(X, r)$ and gives a different $Sp_r(X)$ for each $r \leq 1$ where 1 is the length until feature repetition on the specific axis defined by θ . This r corresponds to the one defined in the LCS algorithm.

One application of the Hough transform is to translate a noisy hand-drawn curve into a b-SPLINE or Bézier curve that interpolates points on the original shape. Since the tilings are regular, the angles between the compositions are always deterministic, and walks on the edges or dual space of a tiling will always produce curves more similar to interpolated ones. Because in a UI it is often the case that the same sequence of operations is repeated, it is proposed that a history lookup based on feature matching can improve the UX.

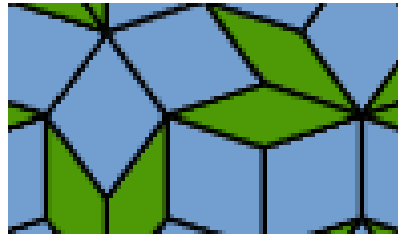
Mathematically the similarity between walks on the dual space of regular tilings, quasi-periodic tilings, and the features extracted from gestures, is given by the Sp_r relation and the LCS description.



(a) The Penrose tiling.

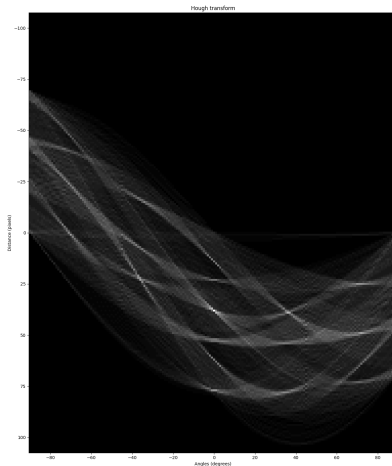


(b) Region 1 of the Penrose tiling.

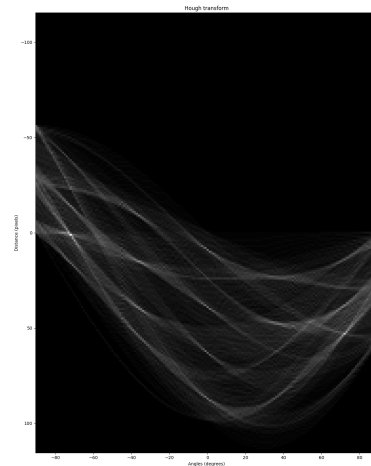


(c) Region 2 of the Penrose tiling.

Figure 3.5: Regions of the Penrose tilings chosen to sample regions that are different. Source: Uploaded by Wikipedia user: Inductiveload using a creative commons license.



(a) Transform of region 1.



(b) Transform of region 2.

Figure 3.6: The Hough transform of the two regions, showing distinct features. Own work using the *scipy* library.

3.4.2 Image retrieval

This section covers, the potential of extending the previous feature based techniques to higher complexity neural networks. Image retrieval is one such task, but is merely

a subset of the larger deep neural models containing "deep features". Image retrieval uses a neural network's hidden layers to perform a matching between one type of visual input such as a drawing, and another, like videos or images. This is quite different from using hand-crafted features in a few ways: the hidden layers can have many nonlinear transformations, and it might not be possible to localize the motivation for the model choosing a particular set of results to a restricted region of its layers. The models might still prove useful because they are able to adapt to very large datasets through the process of reinforcement learning.

Representation learning and graph neural networks are briefly shown to have theoretical potential, but are left unexplored for the scope of this study. In a graph neural network, the layers are connected in well-defined ways, and it is sometimes possible to localize features to particular nodes of the layer graph.

Suppose we have access to a dataset of images $i \in O \subset \mathbb{D}^n$, and we have the function:

$$\begin{aligned} feat : \mathbb{D}^n &\longrightarrow \mathbb{D}^n, \\ feat(X) &= \langle \lambda_j \rangle, j \in \mathbb{N}, \lambda \in \mathbb{D}^n \end{aligned} \tag{3.36}$$

the function of feature extraction, producing an eigenvector of common features in the space \mathbb{D}^n , the space of the image dataset.

The neural network should maximize value i of the function:

$$\mu(X) = \{\forall i \in O \text{ s.t. } \max |feat(X) - feat(i)|_i\} \tag{3.37}$$

which means find the values i that have the most feature similarities with the input X .

The function might be shown to have complicated heuristics even if hand-crafted, because of the fact that the representations of the problems have a dual problem associated with them, both being equally viable.

Chapter 4

Application development

This chapter follows a well structured development methodology. Some founding principles included the ability to interface the framework with other applications by offering a generous level of decoupling. An entire ecosystem exists around digital pen software which is especially developed on Windows. Many commercial SDKs are available on windows only. The Windows Ink component of the operating system can often also extract pen metadata such as pressure and orientation where some SDKs cannot. An open source solution was chosen in the end where all of the dependencies have a free and open public license. This was motivated by the limited number of available drawing applications across all development libraries including .NET and Python. Some applications and libraries that are available do not offer granular control over the canvas or access to internal data structures. For a few reasons it was considered vital that the application exposes pixel-level access to the developer. The classes in the solution offer access to the most fundamental data structures associated with such a hand drawing application with the compromise that some details and subalgorithms had to be manually implemented.

4.1 Requirements

In Chapter 3, the problem domain associated to the application is defined by specifying some of the objects that are represented in memory, and the constraints imposed upon them. Following, a specification of the requirements is created which meets the industry standard requirements engineering practices. After the requirements were identified, application use cases were designed in order to cover all requirements, after which actors and their roles were identified. The purpose and objective of the study and of the software is restated: to create a meta-analysis framework which captures data about the user's interaction with a user interface based on the principles of representation theory. The framework and the study should be applied

to data about user interaction, and interpret it according to the principles mentioned in the mathematical description from the other chapters. The possible scenarios where the designed solution is applicable is both user input and structured layouts that can be represented as planar graphs.

To begin, the requirements are identified in the table below:

REQ1	Map input to category/representation.
REQ2	Map user interface to category/representation.
REQ3	Map input to biometrics.
REQ4	Map biometrics to representation.
REQ5	Information retrieval.

Table 4.1: Requirements table.

And the use cases implemented by the application components are:

UC1	Change scene.
UC2	Information retrieval.
UC3	Insert query augmented by a representation.
UC4	Input curves and shapes.
UC5	Input biometric data.

Table 4.2: Use cases table.

There is a correspondence between each use case and each requirement, which is seen in the figure below:

Requirement	PW	UC1	UC2	UC3	UC4	UC5
REQ2	15	X		X	X	
REQ5	10	X	X			
REQ4	5			X		X
REQ1	10				X	
REQ3	5				X	X

Figure 4.1: Requirements-to-use-cases traceability matrix.

The resulting system is viewn most generally in the following use case diagram:

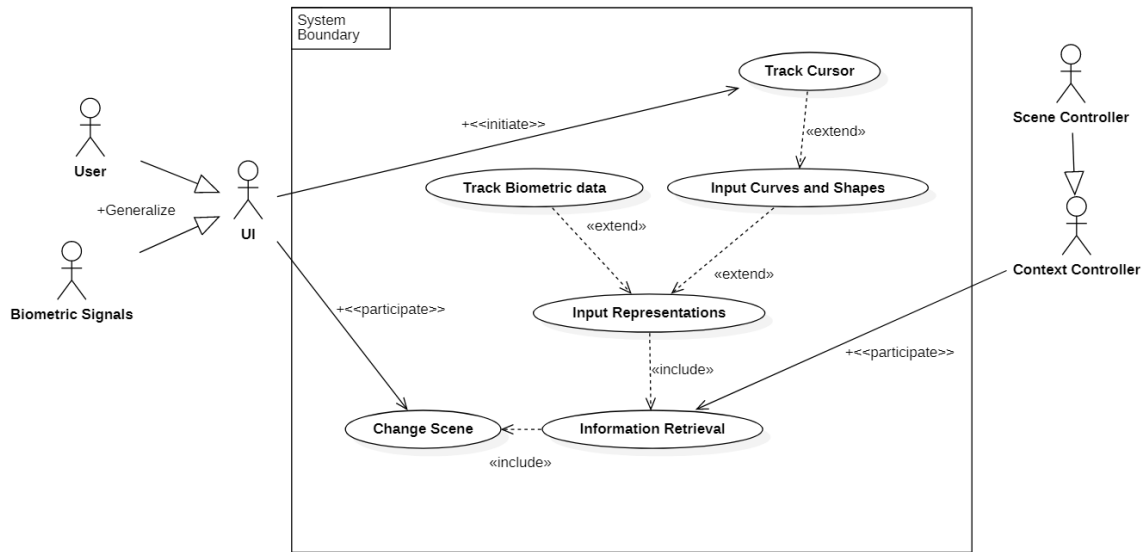


Figure 4.2: Use case diagram.

The use case diagram shows the relations between the use cases. In this case the UI is a generalization of the actor. This highlights that the user is given a level of decoupling from the actual uses of whatever application layout may be analyzed with this framework. The user should interact without constraints and continuously, and be given the ability to perform meta-analysis of his/her own actions, instead of interacting directly with specific application use cases through clicking permanently placed buttons on the layout. The *Context Controller* actor has the task of composing individual input atoms of the meta-UI (the UI component in the diagram), and use the information retrieval engine to couple with any particular application's use cases.

On the other side of the system boundary, the receiving actor or operator must interact with the gesture data and trigger events. Therefore the receiving agent or operator has the role of a mediator, to interpret the user's intentions and change the conditions of the system. A use case has an associated actor to it, which can be seen in Fig. 4.2. By tracing each use case to each requirement, it is also visible which are the cross-cutting concerns. The information retrieval use case "UC2" is a broad concept. The structure of the returned data from the information retrieval module might not be easily predicted because of the nature of the input. Some inputs, shapes, gestures, might retrieve very dissimilar data, and some might retrieve very similar data, for reasons which are known only to the machine learning model that is used at this point. A temporal analysis of this step is seen in the sequence diagram in Fig. 4.3. Otherwise, there is good separation of concerns in the appli-

cation because there are at most 2 use cases attached to each requirement, with the exception of REQ2.

The sequence diagram in Fig. 4.3 shows a combination of scenarios of interaction between the application components. There are two cases to discuss when considering a retrieval model, whether it's a machine learning inference that continuously adapts, or a static evaluation of the input shapes based on feature retrieval augmented with the planar embeddings of the cursor movements on the tiling lattices.

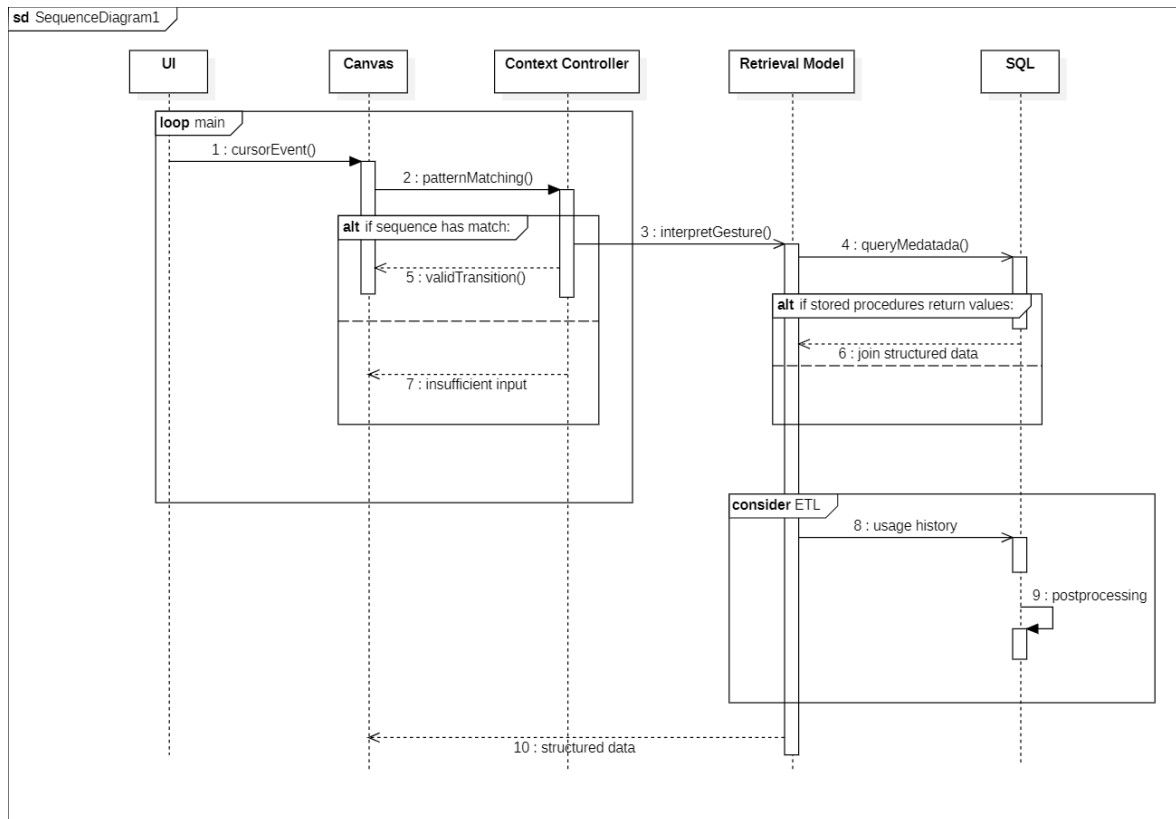


Figure 4.3: Sequence diagram detailing the interaction between the UI and the Controller as well as the SQL component's communication with the machine learning model.

The resulting application can be summarized by the general application diagram in Figure 4.4. The SQL component was intentionally left decoupled from most components. It might be desirable that the data gathered is visualized or some other transformations are done. The SQL component can have stored procedures that perform successive transformations and provide them to a third Python component, which could be responsible for only visualizing. In SQL Server, stored procedures can be created by using Transact-SQL, or by using the common language runtime and one of the Visual Studio programming languages such as Visual Basic or C. An ETL pipeline can therefore be coupled with any type of data understanding application including a web application, a perk of the decoupling offered by the SQL

component visible at the bottom of the figure.

Figure 4.4 contains two instances of Python even though the connection to the SQL consists of the same code. The drawing board app has a duplicate of the SQL Connector because to account for bandwidth limits, a multiprocessing approach was taken instead of a multithreaded one. The *atom* represents TypeScript component as used in React. There are sections of the Neurosity Notion.js API that do not comply with ES6 principles, nor do they use typing, so the *atom* can be replaced by a NodeJS hexagon if desired. The flow of the diagram is top-down with the destination inside the SQL. ETL stands for extract, transform and load, which are stored procedures with the destination inside the same database. A missing Python component on the figure can be considered the data visualization code, which has its own discussion in section 4.4 using the Jupyter notebook.

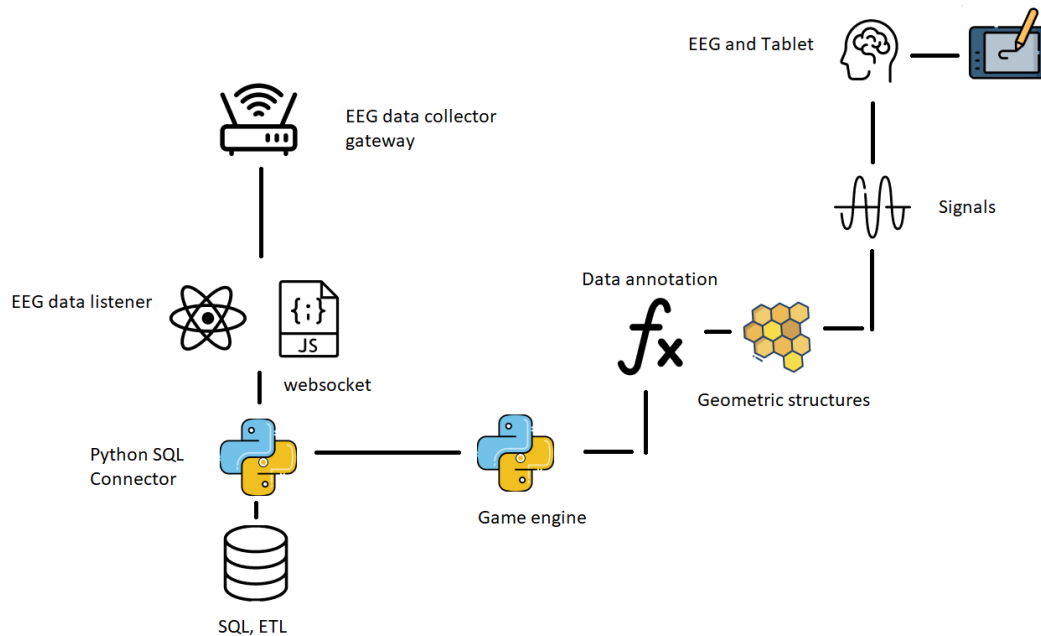


Figure 4.4: Top-down application diagram.

The EEG data listener is a javascript service connecting to the Neurosity Notion.js developer API providing data collection from the cloud of the signals emitted by the electrodes. Using websockets, the Python SQL Connector listens to the javascript service and inputs the data into the corresponding tables in the database. The game engine has its own tables for collecting user interaction data like drawn curves, key presses, changes in the configuration of the session.

4.2 Analysis and design

The analysis and design phase consisted of some case studies and experiments with various operating systems and platforms. It was a clear requirement that the application had to be compatible with digital pen tablets and an EEG headset. In the last subsection, justification is given for the dependencies chosen in accordance with the requirements analysis.

4.2.1 Dependencies and structure

The game is written in Python 2.7 and Python 3.9+ which is separated into at least 5 classes of packages/modules. For more explanation on the reasoning behind this choice review the software alternatives remarks of section 4.2.2. The whiteboard depends on a few external python packages, available from the PyPi registry: `pygame`, `pyodbc`, `numpy`.

A configuration file can be used to define some global parameters about the game logic, the UI, logging, and game modes. Here, the random variables χ_i specified in Section 3.1.3, equation 3.20, are defined as a list of possible values that can be sampled. The game may of course choose to alter the joint probability distribution which affects $P(\chi_i)$, but the values that can be sampled still respect the configuration. For example, when recommending a batch of length k of edges, between each render on the screen the program may wait 100, 200 or 300 milliseconds (ms). This means that for stroke number k_n , it will take a minimum of $t = n \cdot 100$ ms, and a maximum of $t = n \cdot 300$ ms. The more often the sampling $P(\chi_i)$ occurs, the higher the probability of having a non-minimal time is, within n rolls.

The main class is the *Program* class, which captures events from the system using *pygame* utilities, and passes them down to the member classes. Among the member classes there are the following: *Whiteboard*, *Tiling*, *Paintbrush*. They respectively use lower level concepts such as custom classes including *Stroke*, *Line*, *Path*, and *pygame* classes including *Rect*, *PixelArray*, *Surface*.

Note that these game objects are fundamental enough for even a simple feature extractor to be used to make a relation between other game environments and user interfaces to this framework's abstract GUI formulation. Convolution layers in convolutional neural networks act as feature extractors themselves. Deep layers of a neural network especially encoder-decoder architectures have high level features connected to the labels of the data in order to synthesize new previously unseen data. If acting on a set of simple features like lines, curves, polynomials, the networks weights might be more easy to interpret when using tools for visualizing them.

The framework also contains a *shape recognizer* neural network model, also written in Python, that can use an interval of strokes $s_i \in S$ to recognize specific shapes which are in turn detected by the game by periodically querying the DB for events or receiving an event notification through a web socket.

Another decoupled software module is the *point cloud extractor* which provides a tiling to the game. The point cloud extractor is written in Python, and parses an SVG file to produce a set of edges formed by points in the tiling. The SVG contains absolute and relative coordinates of the endpoints of edges of vectors. The tilings used are preiodic or quasi-periodic regular tilings, made of fundamental regular polygons, like triangles, pentagons, squares, etc. Even though SVG supports rendering of curves, there is no tiling containing curves used by the framework. The reason is that for quasi-periodic tilings, curves might be given by a mapping of walks on the edges using a weighted average. A similar case was discussed in 3.2 when defining the LCS algorithm. Consider also the interpolation function $\theta(p_{i,j})$. Now the average

$$\phi^3(s_i) = \frac{\overline{xy}_{s_{i1}} + \overline{xy}_{s_{i2}}}{2}, \quad (4.1)$$

is given to obtain $\theta(\phi^3)$, the interpolated averages, which are curves. When applying this to quasi-periodic tilings, one has to start from a point p and see what is

$$|spec(\phi^2)|_{r'}, \quad (4.2)$$

where

$$d(p, r') = \sqrt{\sum (p_{\overline{xy}} - r'_{\overline{xy}})^2}, \quad (4.3)$$

holds. The curves in the spectrum set is another case of the exemplified $Sp(X)$, which could be an indicator of the complexity of a tiling or of a stroke history which has an intersection with a tiling. This will allow for the *feature extractor* of the gestures to be connected with the complexity analyses, which won't be detailed in this study but can be expanded upon. One can imagine that UI transitions can be changed using the parametrized components to obtain a set of curves that is as varied as possible, in order to not accidentally perform the wrong sequence of clicks and fail to reach the desired transition.

4.2.2 Motivation for software choices and alternatives

The software was tested with external hardware such as a Wacom pen tablet on Windows 10. This is not a requirement but was in interesting case study because Windows provides integration using Windows Ink, which is a service that connects the operating system to the driver. This allows pressure to be forwarded to an application using the SDK and various gestures to be performed when using the OS

itself. The most valuable data were described to be the changes in complexity associated with the transitions, and possibly kinematic data. Because the sampled points are timestamped, one can measure the velocity of the drawing strokes, but can also omit them when not relevant. A pen only changes the kinematics of the user, and not the drawing necessarily. If whole body tracking was included then more complex gestures could be performed, but it remains to be determined if the data would be relevant to the Euclidean embedding of regular tilings. It might be possible to investigate whether hyperbolic tilings work better in 3-dimensional space, or if whole-body kinematic can correspond to higher-dimensional tilings. Since the higher-dimensional tilings have a 2-dimensional analog the results could still be correlated with the findings of this study, as some 3-dimensional space filling tilings are obtained by mere projections, rotations and reflections of their two-dimensional building blocks.

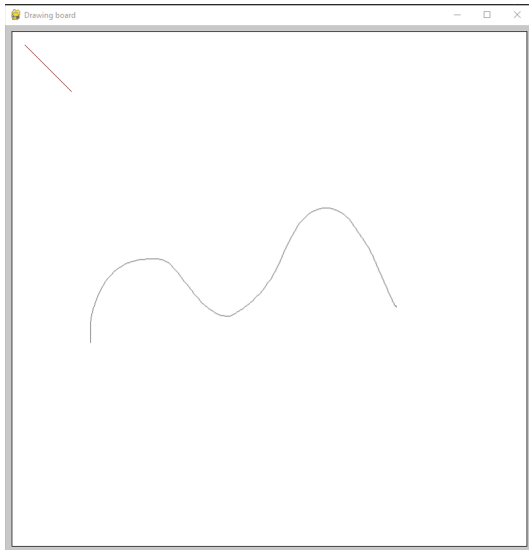
To keep development integrated in the Windows ecosystem, it would have been preferable to use C, with .NET's legacy classes for drawing. The *System.Drawing* namespace can be used with legacy windows forms apps. Newer WPF apps, in which the UI is specified by XAML, have the *InkCanvas* control. It was not possible to choose this for the framework because there is no control over each individually sampled point over which the cursor hovers. The .NET WPF app would also interpolate the points with their own methods, instead of the chosen Bresenham's line algorithm. There is no also way to control the sampling speed and therefore the number of points chosen for interpolation. The framework's Python 2.7 solution which allows for pixel-level manipulation was chosen despite losing a signal modality, that of the pen's pressure, which was easier to integrate in the Windows ecosystem.

An alternative for the drawing board that was considered is also a JavaScript native environment. D3.js allows for manipulation of an SVG element to produce animations shapes and drawings. The problem was that accessing the logic behind the sampling and interpolation was again difficult. The SVG element is a HTML native control, and the JavaScript low level routines for pixel-by-pixel access were not visible. Also, when using React together with the previous, there would be a conflict of access over the base SVG element that corresponded to the canvas. Still, SVG would have had some nice properties. First of all, it is based on XML, and there is a 1-to-1 correspondence between the HTML SVG tag and an SVG image. A minus is the complexity of the internals and high number of particularities to various operating systems and browsers. An hierarchical representation such as XML and SVG can still be one of the best ways, because they can be modelled as nodes of a tree or graph. The nodes can be annotated with custom attributes and also geometrically embedded and reorganized. Many classical problems in Graph

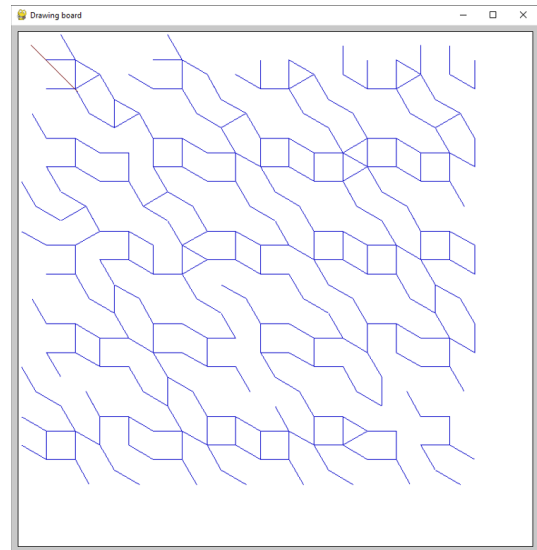
theory are concerned with these aspects and are closely related to the mathematical discussion of the correspondence between the tilings and the notion of complexity with respect to their representations.

4.3 User interface and game window

The data collection app is presented as a game where the task is to complete the current pattern. Patterns viable for the game are plane-filling tilings of shapes. The choice of space-filling tilings and curves is based on the pre-existing wealth of literature describing the representation of these geometric structures. The algebras underlying their representations can not only be used to identify and isolate stimuli but also to generalize user interfaces and prove similarities between transformations of the same object. Tilings can be described by finite groups, or finite reflection groups to be more specific. If a system can be proven to have a sufficiently close graph representation, for example by using a specification language that can describe hierarchies of objects, then by using rewriting rules acting on the graph the resulting graphs might be objects that belong to the same group. Since the hypotheses of the framework's knowledge mining system are based on algebraic descriptions of the fundamental components of the system, we can look for patterns even in very complex systems by rewriting them to an equivalent but simpler one. For a detailed analysis refer to the next section about tilings.



(a) A brush stroke drawn by the user.



(b) A partially completed tiling made of game predictions.

Figure 4.5: The main window of the drawing board application, with a reference diagonal line in the upper left corner.

The game loop has two modes to best capture both the tasks of kinematic data

collection and generalized user interfaces: a mode where the task is to draw a set of tilings based on sequences of strokes, and a task that simulates a user interface and the transitions between the activities meanwhile capturing the cursor's position.

To help achieve the goal of drawing the tiling, the game engine recommends a series of next strokes, highlighted on the screen, based on complex logic. The player then draws over these recommendations.

Deviating from the recommendation is not a problem, but a valuable stream of information. The game should detect if the user's initial intention is one of the best possible recommendations in the first place. If it is not, the game can adapt the next recommendations to better fit the current user's inputs and adapt while transitioning to a good outcome. The latter should be considered a human-in-the-loop (HIL) feedback system, because while the system adapts, the user is also learning and becoming habituated. In the case of the HIL system, the learned behavioral patterns should match one of the best possible combinations, as well as be interpretable by other users.

The player has the choice to reset the game or finish the current tiling. The game engine decides when the input is sufficiently close to the goal so that the task is considered complete and after which the player has the option of transitioning to the next task.

The game is allowed to choose the next task based on the previous tasks, but the detail of introspection of the previous tasks is limited to general statistical indicators. For example, the current task should not depend on fine-grained data of the previous task but should instead depend on the previous task choices and some indicators of the complexity of the previous task.

The main window, as seen in Fig. 4.5, has no buttons. The way to navigate is by drawing a gesture, which is fed into the shape recognizer artificial neural network.

4.4 Database

Microsoft SQL Server 2019 is deployed on an Intel i7 machine with a 256GB SSD to collect metadata about the game as well as sensor data. The database was tested with fast time series data, of a bandwidth of at least approaching 64kbps. The database ran without performance bottlenecks for at least 180 seconds per game session. Indexing was attempted with respect to the timestamp column of the table, but there were some issues when doing this on 50000 records. A better DB alternative such as Redis, which is well suited for time-series data, was left unexplored.

The database is structured in tables that are mainly used for logging. A log of strokes exist at runtime that is replicated in the DB. Additional environment meta-data such as game mode selections exists. The gestures are logged in a separate

table. The strokes themselves constitute a time series, and every metadata is timestamped, but additional tables can be included for time series data. As previously mentioned, the *MS SQL Server 2019* is capable of handling a few minutes of synchronous time series queries including insertions. For a discussion on processing the time series tables review Chapter 4, as more details are presented in a comparison with image processing and video processing with machine learning techniques.

The ETL Pipelines consists of stored procedures that are triggered synchronously by game events and run asynchronously on *MS SQL Server 2019*'s engine. These are transformations applied to tables and could be thought of as functional programming designs. The source table is not modified but an output table is generated with the results, which are then updated on re-running the query, making the stored procedures pure functions. The output tables are manually specified into the configuration of the application and are queried by any of the other software components. This level of decoupling introduced by SQL Server's modularity and asynchronous processing is worth the processing power even for a small application. Stored procedures include methods for querying past predictions with constraints, past user input, optionally sequence matching algorithms, and lastly generating human-readable statistics about the previous sessions.

4.4.1 Data analysis

The purpose of the framework is to collect, annotate and correlate data, but not to conduct any particular controlled experiment and draw conclusions on that data. Despite this, some data analysis discussions are provided in the repository. The recommended way to interact with the SQL to perform data analysis is using Jupyter notebook with an SQL connector. An experiment was conducted on PostgreSQL as well as MS SQL Server 2018 to visualize interesting aspects of the collectable data.

Jupyter notebook is a python package using IPython that can execute Python code cells individually and display inline graphics such as data visualizations, plots, spectrograms. Choosing Jupyter simplifies deployment because it reuses the Python dependency within the framework. The ETL pipeline does not include transformations of the data that generate spectrograms, because storing images within the DB was not a design objective. The responsibility of generating visualizations of the data falls entirely on Jupyter notebook.

For visualization the *matplotlib*, *scipy*, *numpy* are used throughout the projects. To cluster a set of time windows a possible approach is to collect spectral density information about the alpha, beta, gamma, delta power bands of the same time window and compute similarity by difference in amplitude. An example of a randomly selected time window can be seen in Fig. 4.6.

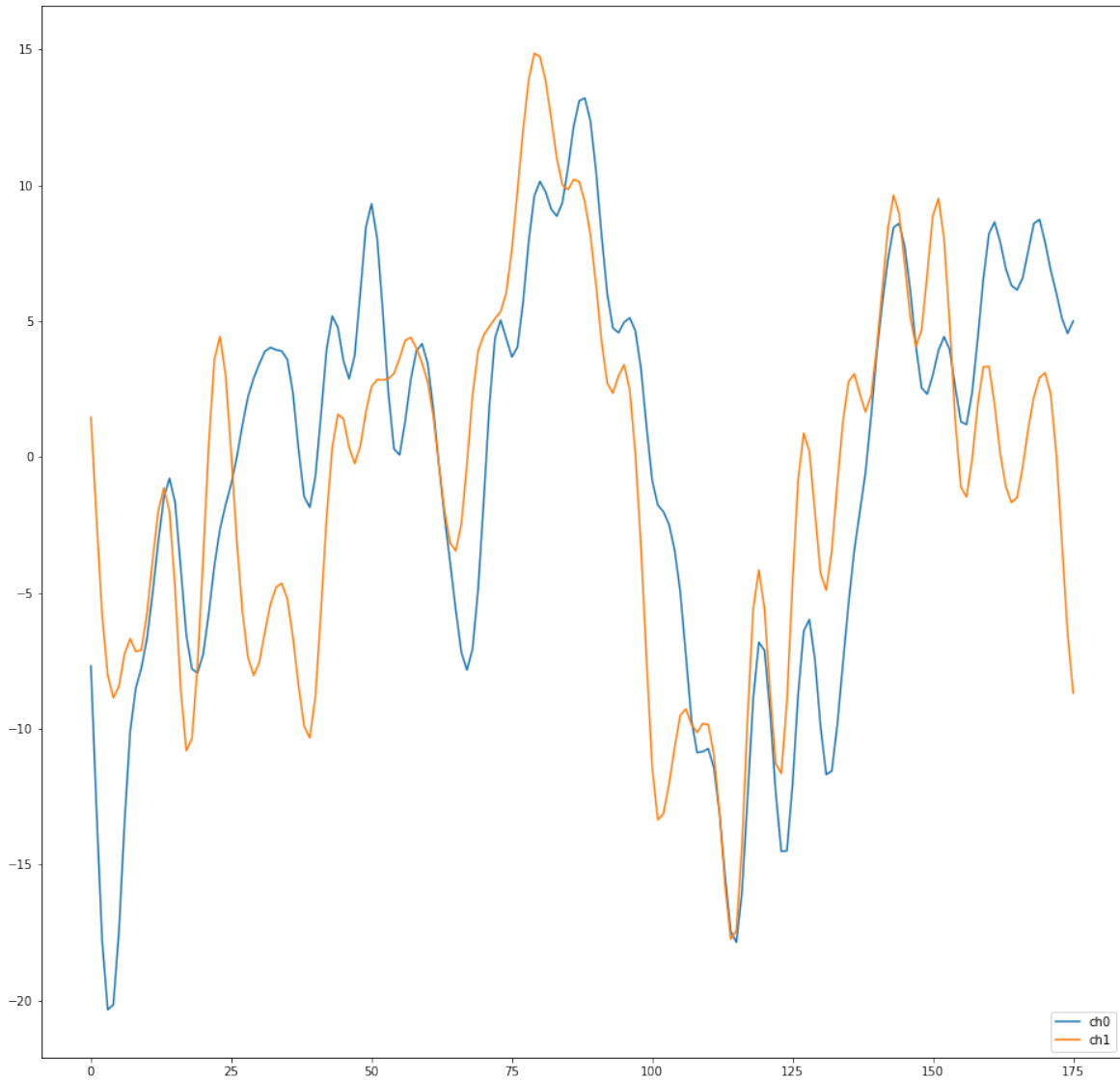


Figure 4.6: An example of a 175ms time window of EEG data visualized from the IPython kernel.

An expert might interpret the categorized data in order to build an automated system for optimizing the game recommendations based on the biometric data offered by the user. In the sequence diagram in Fig. 4.3, the ETL serves the purpose of continuously generating insights that are of interest to the data analyst expert. SQL triggers offer the mechanism by which stored procedures are invoked by particular inserts, or otherwise can be explicitly invoked by the application through executing a query. In the use case diagram, the *Context Controller* can be seen as the expert system adapting the scene in order to inform the user about past decisions, the scene, the ways in which the scene might change, etc. A concrete example of a stored procedure that an expert system might employ is using inter-hemispheric synchronization. The EEG electrodes are labelled by spatial position on the scalp. If two signals are considered similar on symmetric electrodes situated on different hemispheres,

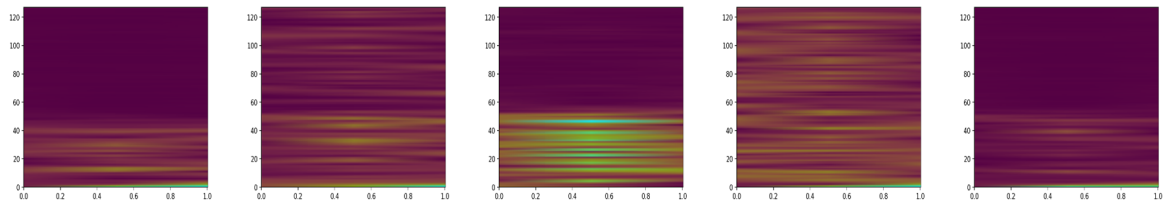


Figure 4.7: A set of 5 spectrograms from different channels concatenated for experimenting with neural networks. Computer vision neural networks are less likely to classify such images with classical architectures, where a 1d representation is simpler to implement and use as input to existing models.

for example T5 and T6, by comparing the spectral power amplitude on a frequency band, the game might choose a future behavior that accounts for this. If otherwise the inter-hemispheric synchronization is the desired outcome, then the game engine could interact in such a way that could evoke a response in one of the hemispheres which could induce a synchronization. In Fig. 4.7 there are 5 STFT spectrograms, one for each of 5 randomly selected channels, where a data analyst might be able to visually compare the similarity of two channels.

Chapter 5

Human-computer interaction and virtual reality

The possibility of gathering other types of signals from a human agent has been left out so far, because it is a separate concern. The framework defined should be versatile enough to collect important information to be used in studies without any biometric data, because the geometric structures in certain game tasks were chosen to be represented by algebras describing the data. There exist many open source repositories containing hierarchical descriptions of user interfaces using annotated XML files that can be used for experimenting with layouts given that a Coxeter group can be associated with them. As is briefly mentioned in the concluding remarks in Chapter 6, there are possible applications using the formal language techniques for users familiar with American sign language, especially in VR. Eye tracking is also an input homologous to pointer tracking, and can be useful in certain medical contexts, and is already harnessed by VR headsets. The study's argument might be then that universal representations can be perceived by the user regardless of the channel through which they are expressed.

5.1 Measurements related to brain activity

5.1.1 Background

To date there is no clear localization of emotion in the brain. Paul Ekman and Klaus Scherer provide an elaborate discussion of this topic in their works. Valence and arousal are commonly used to refer to the subjective appearance of emotions and feelings. The theory of appraisal is an important driver of the processing of an emotion. [27, 28] In the introduction there were already comparisons made between nonlinear systems and the human brain.

The theory of appraisal considers the subject behaviorally, to avoid the exploding complexity of understanding every aspect of the human brain. Statistical correlations are still observable with respect to the subject's emotional states and the environment's dynamics, and can be usefully applied without requiring complete determinism. Russel's valence arousal model is another very powerful model in conjunction with EEG measurements, and can complement the theory of appraisal. Classical experiments of measuring emotional response can be seen in Maruyama et al. who performed an independent component analysis with a source localization method, to obtain distinct neural activities from EEG signals. They propose that neural activity can correlate with specific emotional states, depending on the frequency and location of the signal. [29]

Traumatic brain injury can have different symptoms depending on the area affected. Classically some areas have been identified and named according to commonalities between medical subjects. The problem of determining the explanation of the symptoms is still complicated. Good communication between brain regions involved in specific tasks is important to consider when recording neural activity. Brain regions are not only connected by their proximity on the cortical surface, but also by projection fibers, efferent and afferent fibers that bridge cortical regions by traversing the depths of the whole brain. These fibers can be identified using fMRI, which produces tractographies. Thompson et. al have shown using diffusion tractography that impaired communication between the motor and somatosensory homunculus is associated with poor manual dexterity. [30] Forming hypotheses about what consequences a traumatic brain injury can have is not always evident, because the interactions of the brain organoids are conditionally dependent. It could take a human a very long time to form a theory based on existing evidence, and for this reason automated proof assistants might be employed. The previous chapters have shown a mathematical framework for correlating measurements about kinematic motions on a user interface. Correlating the gathered information with synchronous biometric measurements could prove useful to form complex hypotheses about physiological profiles of each user.

Kropf et al. have conducted a review about the involvement of the somatosensory cortex in processing emotion and empathy. The primary somatosensory cortex (SI) and the secondary somatosensory cortex are both found to be involved in tactile attention. [31] The generality of the kinematic data on a 2-dimensional surface is valuable because it can take many forms. Eye tracking is also kinematic data, and produces a tactile sensation. Moving one's head also produces a different kind of sensation from the balance centers located in the ear.

Studies from immersive environments such as virtual reality [32] have suggested that the cortical homunculus is capable of adapting to novel bodies. Homuncular

flexibility is a paradigm in which physical motions are transformed by remapping degrees of freedom from tracked movements onto an avatar. VR interventions for phantom limb pain has been shown to give significant reduction in symptoms. [33] The mechanism by which altering the visual input of a patient suffering from phantom limb pain reduces pain has been proposed to be a cortical reorganization, enabled by neuroplasticity. Prolongued exposure to VR environments could produce phantom sensations and synesthesia which might not have an easy explanation, and for correlating them a biological model of the brain could be used in tandem with the kinematic data by the proof assistant. The tactile sensations seen in Kropf et al. might be generalizable across different sensing organs, when considering such changes of embodiment. An example of a connectogram that might be used to draw biologically inspired hypotheses is seen in Fig. 5.1 which references a functional specialization study by Yeo et al. using fMRI and data about the human cortex. [9]

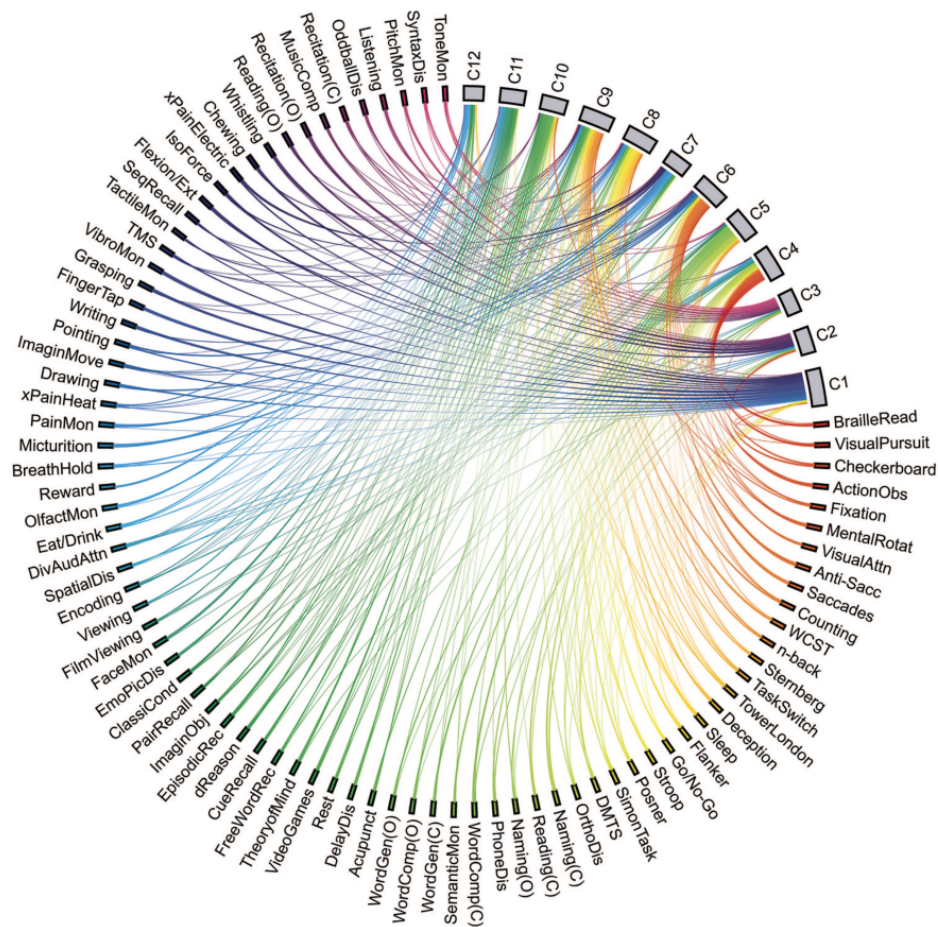


Figure 5.1: Visualization of the brain recruiting a cognitive component during a particular task as defined by Yeo et al. performed using the Circos software. [9]

5.2 Data and software

MNE is a Python framework for working with EEG data. [34]. Several spectrograms were generated with this framework, and it was useful because it incorporates many of the file formats used for time series data, which have capabilities such as annotating sections of a signal. Other time series plots and spectrograms were done directly in the Jupyter kernel. A complete list of packages is found in the data and software availability sections of each corresponding chapter. Other packages used were *scikit* for generating STFT spectrograms.

The scope of the framework presented does not go beyond collecting and correlating the data. Even though advanced classification could be made and even prediction and simulation of signals together with simulated pen strokes for testing, it goes beyond the scope of the study. Some authors have published methods for generating most likely EEG signal given a time window, as seen in TimeGAN. TimeGAN is a generative model that can learn high dimensional relationships of the signal dynamics to generate realistic signals. [35] This architecture has been adapted in the past for privacy-preserving medical data generation, as well as EEG signal generation. [36, 37] It has been shown to work on large datasets and could be a starting point for a future direction.

The neural networks used for classifying signals are trained using Tensorflow. An interesting problem arose when classifying multi-channel data. The problem is that neural networks are not suited for n-dimensional inputs, which is immediately obvious from the API's design. Some neural architectures capable of classifying n-dimensional inputs were found in the literature, but often perform poorly. The experiments performed for this study are specifically using a spectrogram to compress the channels to a 3-dimensional image form, which the neural network can accept. One attempt of inputting a transformation that achieves a 1-dimensional input was studied, yet the more relevant case is not choosing a time window linearly but creating a graph pairing between EEG signals and the intervals between the pen stroke endpoints. This is directly compatible to the dual space discussion in Section 3. The application does not have performance issues in a 3 minute measurement session, and if this is a problem the SQL server can run the stored procedures on more capable specialized hardware.

The pen strokes are used to generate a set of most likely EEG patterns to follow, in a multi stage pipeline, which detects the likelihood and frequency of spikes or ERPs, predicts a likelihood of abnormalities, estimates fatigue and cognitive performance. Depending on the type of end to end system that is expected, the architecture of the multi stage model could be normalized, or split into smaller independent models. The pipeline can be of course adjusted to match the probability distribution found

in Section 3.

There are trained neural networks available for the gesture recognition functionality which consists of a CNN for classification of hand drawings. Feature extractors are also used. The hand gesture recognition can be trained with alternative representations of the strokes including features extracted from the strokes using the Hough transform.

5.3 Data and code availability

There was no data collected for the study except for test data that was removed as soon as it was collected. The neural network models and feature extractors are available on the GitHub repository. There are available Python modules which perform data analysis tasks that were used for case studies but were not part of the final solution. The code related to the whiteboard application is fully available as well as the EEG signal listener pipeline.

Code specific to this chapter are modules that process signals, utilities for plotting signals, Jupyter notebooks with experiments. The respective code is available on the repository at <https://github.com/p0licat/thesis-eeg> which also includes code from the previous chapters.

Chapter 6

Conclusions

The main components of a general UI meta-analysis framework was presented. It was shown that a simple 2-dimensional drawing board type interface is general enough to discuss universally applicable user interface concepts. There is no loss of generality by collecting data about solving drawing tasks because the choice of the drawing objective in this case is very abstract and simple: the periodic and aperiodic tiling. The layout of the tilings depends on the rules used for generating them. The rules for generating them include rotation, translation and mirroring operations, which are geometric transformations. They have a concise and clear mathematical description, and produce results that are easy to categorize.

The study presented details about the compressed mathematical representations that constitute tilings. A connection was then made between the representation of tilings and that of user interfaces. A connection is also made between those two and the patterns generated by a user interacting with them. A user's navigation through a user interface produces kinematic data that is tracked, and forms structures similar to the tilings, with their own measure of complexity. The connection between tilings and user tracking is applied to gesture tracking, by integrating gesture detection directly into the application while removing any other way to interact such as clicking. The aim of the motion tracking module is to show that the 2-dimensional environment can be translated to more general concepts and high dimensional structures with the help of the mathematical language that describes the tilings.

The parameter adjustment of the drawing tasks feeds back continuously to the user input. By making remarks about interactions with 2-dimensional environments, more generalized conclusions can be drawn and applied to higher dimensional environments like VR and AR, or other scopes like phonetic systems in voice recognition, in principle. Any system that can form a 1-to-1 correspondence between a mode of user input and a particular subset of strokes on a canvas can theoretically be translated to these conclusions. To do this, the algebra corresponding to that set of stroke's particular class is used to look up their higher dimensional

counterparts. This is possible because regular tilings, regardless of their periodicity, are made of fundamental polygons, which have higher dimensional analogs with similar properties, for example the platonic solids in 3 dimensions.

The concluding remark is that the number of directions in which this can be explored is potentially very large. The mathematical language used to analyze the presented topics should not be disregarded when designing all aspects of the user interface, be they layout design or input modes, or transitions between elements, or generalizability towards higher dimensional environments.

6.1 Future work

Compositions of tilings for specifying the matching layouts or for composing queries to interact with a UI using feature extractors could be expressed more easily in VR where the user can use both hands for gestures, and the 2D panes that are embedded in the 3D space themselves have a unique layout. In VR headsets you can often already use your voice to perform queries which is a sort of feature extractor in the sense that the model understands the voice signal and retrieves appropriate results. Virtually any number of connections can be made when the system is specified and understood at the level of grammars and proper validation occurs in the development pipeline. American Sign Language (ASL) could be well fitted for VR tiling based user interfaces and data embeddings. Using the annotated representations of any geometric structures can be a tool to provide accessibility to users who are better compatible with certain communication channels. Representationally annotated user interactions could be used to study how the user learns to adapt to an environment and to understand the data that is embodied there regardless of the preferred method of interaction (auditive, visual, tactile).

Bibliography

- [1] Masafumi Oizumi, Larissa Albantakis, and Giulio Tononi. “From the Phenomenology to the Mechanisms of Consciousness: Integrated Information Theory 3.0”. In: *PLOS Computational Biology* 10.5 (2014), pp. 1–25. DOI: 10.1371/journal.pcbi.1003588.
- [2] Matlab / Mathworks website. “Sensor Fusion and Tracking Toolbox”. Last accessed July 2022. URL: <https://www.mathworks.com/products/sensor-fusion-and-tracking.html>.
- [3] Keigo Tada, Ryosuke Yamanishi, and Shohei Kato. “Interactive Music Recommendation System for Adapting Personal Affection: IMRAPA”. In: *Entertainment Computing - ICEC 2012*. Ed. by Marc Herrlich, Rainer Malaka, and Maic Masuch. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 417–420. ISBN: 978-3-642-33542-6.
- [4] Murat Üney, Bernard Mulgrew, and Daniel E. Clark. “A Cooperative Approach to Sensor Localisation in Distributed Fusion Networks”. In: *IEEE Transactions on Signal Processing* 64.5 (2016), pp. 1187–1199. DOI: 10.1109/TSP.2015.2493981.
- [5] Eva Ose Askvik, F. R. (Ruud) van der Weel, and Audrey L. H. van der Meer. “The Importance of Cursive Handwriting Over Typewriting for Learning in the Classroom: A High-Density EEG Study of 12-Year-Old Children and Young Adults”. In: *Frontiers in Psychology* 11 (2020), p. 1810. ISSN: 1664-1078. DOI: 10.3389/fpsyg.2020.01810.
- [6] Brian Fiani et al. “An Examination of Prospective Uses and Future Directions of Neuralink: The Brain-Machine Interface”. In: *Cureus* (2021). ISSN: 2168-8184. DOI: 10.7759/cureus.14192.
- [7] Gopala Krishna Anumanchipalli, Josh Chartier, and Edward F. Chang. “Speech synthesis from neural decoding of spoken sentences”. In: *Nature* 568 (2019), pp. 493–498.

- [8] Boring Matthew J. Robinson Amanda K. Venkatesh Praveen. "Very high density EEG elucidates spatiotemporal aspects of early visual processing". In: *Scientific Reports* 7 (2017). ISSN: 2045-2322. DOI: 10.1038/s41598-017-16377-3.
- [9] B. T. Thomas Yeo et al. "Functional Specialization and Flexibility in Human Association Cortex". In: *Cerebral Cortex* 25.10 (2014), pp. 3654–3672. ISSN: 1047-3211. DOI: 10.1093/cercor/bhu217.
- [10] Dileep George and Jeff Hawkins. "Towards a Mathematical Theory of Cortical Micro-circuits". In: *PLOS Computational Biology* 5.10 (2009), pp. 1–26. DOI: 10.1371/journal.pcbi.1000532.
- [11] Stéphane Mallat. "A Theory for Multiresolution Signal Decomposition: The Wavelet Representation". In: *IEEE Trans. Pattern Anal. Mach. Intell.* 11 (1989), pp. 674–693.
- [12] Emre Aksan et al. "CoSE: Compositional Stroke Embeddings". In: *Advances in Neural Information Processing Systems 2020* (2020).
- [13] Tomohide Shibata, Daisuke Kawahara, and Sadao Kurohashi. "Neural Network-Based Model for Japanese Predicate Argument Structure Analysis". In: *ACL*. 2016.
- [14] Hajime Morita, Daisuke Kawahara, and Sadao Kurohashi. "Morphological Analysis for Unsegmented Languages using Recurrent Neural Network Language Model". In: *EMNLP*. 2015.
- [15] Ramprasaath R. Selvaraju et al. "Grad-CAM: Why did you say that? Visual Explanations from Deep Networks via Gradient-based Localization". In: *CoRR* abs/1610.02391 (2016). URL: <http://arxiv.org/abs/1610.02391>.
- [16] Roberto D. Pascual-Marqui. "Standardized low-resolution brain electromagnetic tomography (sLORETA): technical details." In: *Methods and findings in experimental and clinical pharmacology* 24 Suppl D (2002), pp. 5–12.
- [17] Haidar Khan et al. "Focal onset seizure prediction using convolutional networks". In: *IEEE Transactions on Biomedical Engineering* 65.9 (2017), pp. 2109–2118.
- [18] Yuki Hagiwara U. Rajendra Acharya Shu Lih Oh et al. "Automated EEG-based screening of depression using deep convolutional neural network". In: *Computer methods and programs in biomedicine* 161 (2018), pp. 103–113.

- [19] Nadia Mammone, Cosimo Ieracitano, and Francesco Carlo Morabito. "A deep CNN approach to decode motor preparation of upper limbs from time-frequency maps of EEG signals at source level". In: *Neural networks : the official journal of the International Neural Network Society* 124 (2020), pp. 357–372.
- [20] Francesco Rundo et al. "An Innovative Deep Learning Algorithm for Drowsiness Detection from EEG Signal". In: *Computation* 7.1 (2019). ISSN: 2079-3197. DOI: 10.3390/computation7010013.
- [21] Hirokazu Takahashi et al. "Convolutional neural network with autoencoder-assisted multiclass labelling for seizure detection based on scalp electroencephalography". In: *Computers in Biology and Medicine* 125 (2020), p. 104016. ISSN: 0010-4825. DOI: <https://doi.org/10.1016/j.combiomed.2020.104016>.
- [22] Garrett Honke et al. "Representation learning for improved interpretability and classification accuracy of clinical factors from EEG". In: *International Conference on Learning Representations*. 2021. URL: <https://openreview.net/forum?id=TVjLza1t4hI>.
- [23] Andac Demir et al. "EEG-GNN: Graph Neural Networks for Classification of Electroencephalogram (EEG) Signals". In: *CoRR abs/2106.09135* (2021). arXiv: 2106.09135. URL: <https://arxiv.org/abs/2106.09135>.
- [24] Valentin Gomez-Jauregui et al. "GomJau-Hogg's Notation for Automatic Generation of k-Uniform Tessellations with ANTWERP v3.0". In: *Symmetry* 13.12 (2021). ISSN: 2073-8994. DOI: 10.3390/sym13122376. URL: <https://www.mdpi.com/2073-8994/13/12/2376>.
- [25] Branko Grünbaum and Geoffrey C. Shephard. "Tilings by Regular Polygons". In: *Mathematics Magazine* 50.5 (1977), pp. 227–247. DOI: 10.1080/0025570X.1977.11976655.
- [26] H. S. M. Coxeter. "The Complete Enumeration of Finite Groups of the Form $R_i^2 = (R_i R_j)^{k_{ij}} = 1$ ". In: *Journal of the London Mathematical Society* s1-10.1 (1935), pp. 21–25. DOI: <https://doi.org/10.1112/jlms/s1-10.37.21>.
- [27] Klaus R. Scherer. "What are emotions? And how can they be measured?" In: *Social Science Information* 44.4 (2005), pp. 695–729. ISSN: 0539 0184.
- [28] Paul Ekman. "An argument for basic emotions". In: *Cognition and Emotion* 6.3-4 (1992), pp. 169–200. DOI: 10.1080/02699939208411068.
- [29] Yasuhisa Maruyama et al. "Independent Components of EEG Activity Correlating with Emotional State". In: *Brain Sciences* 10 (2020), p. 669. DOI: 10.3390/brainsci10100669.

- [30] Abigail Thompson et al. "Impaired Communication Between the Motor and Somatosensory Homunculus Is Associated With Poor Manual Dexterity in Autism Spectrum Disorder". In: *Biological Psychiatry* 81 (2016). DOI: 10.1016/j.biopsych.2016.06.020.
- [31] Erika Kropf et al. "From anatomy to function: the role of the somatosensory cortex in emotional regulation". In: *Revista Brasileira de Psiquiatria* 41 (2019), pp. 261–269.
- [32] Andrea Stevenson Won et al. "Homuncular Flexibility in Virtual Reality". In: *Journal of Computer-Mediated Communication* 20.3 (2015), pp. 241–259. ISSN: 1083-6101. DOI: 10.1111/jcc4.12107.
- [33] Thomas Rutledge et al. "A Virtual Reality Intervention for the Treatment of Phantom Limb Pain: Development and Feasibility Results". In: *Pain Medicine* 20.10 (2019), pp. 2051–2059. ISSN: 1526-2375. DOI: 10.1093/pm/pnz121.
- [34] Alexandre Gramfort et al. "MEG and EEG Data Analysis with MNE-Python". In: *Frontiers in Neuroscience* 7.267 (2013), pp. 1–13. DOI: 10.3389/fnins.2013.00267.
- [35] Jinsung Yoon, Daniel Jarrett, and Mihaela van der Schaar. "Time-series Generative Adversarial Networks". In: *33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada*. 32 (2019). Ed. by H. Wallach et al.
- [36] Cristóbal Esteban, Stephanie L. Hyland, and Gunnar Rätsch. *Real-valued (Medical) Time Series Generation with Recurrent Conditional GANs*. 2017. arXiv: 1706.02633 [stat.ML].
- [37] Kay Gregor Hartmann, Robin Tibor Schirrmeister, and Tonio Ball. *EEG-GAN: Generative adversarial networks for electroencephalographic (EEG) brain signals*. 2018. arXiv: 1806.01875 [eess.SP].